



## EstEID v. 3.5

### Estonian Electronic ID card application specification

Document information	
Initial Release Date	21.05.2013
Recipient	Police and Border Guard Board, Republic of Estonia
Publisher	Republic of Estonia Information System Authority
Author	Republic of Estonia Information System Authority
Version	1.30

Table for Document Version History		
Date	Version	Changes/notices
03.07.2012	0.1	Draft and document barebones
24.07.2012	0.2	APDU commands and responses
31.07.2012	0.3	Card application objects, their details and started with general operations
02.08.2012	0.4	Card application general operations
21.08.2012	0.5	Card application management operations and APPENDIX with APDU log.
10.09.2012	0.6	Clarifications
03.10.2012	0.7	<ul style="list-style-type: none"><li>Removed chapter <u>T0 and T1 protocols</u></li><li>Improved chapter <u>0 Card application file system structure</u></li><li>Grammar and wording corrections</li><li>Few procedure parameters corrections</li><li>Added references</li><li>Changed name of chapter <u>Decrypting session key</u> to <u>Decrypting public key encrypted data</u> and improved given chapter</li><li>Clarified using of RSA exponent in case of different JavaCard platforms</li></ul>
21.01.2013	0.8	<ul style="list-style-type: none"><li>Added chapters <u>1.3 Identifying the card application</u>, <u>7.1 Card possible response in case of protocol T0</u> and <u>7.2.4 GET RESPONSE</u>.</li><li>Grammar and wording corrections</li><li>Improved examples</li><li>Fixed secret keys reference numbers</li></ul>
13.02.2013	0.81	<ul style="list-style-type: none"><li>SSL is obsolete. Changed to TLS.</li><li>Changed key reference identifiers to be backward compatible</li></ul>



Table for Document Version History		
Date	Version	Changes/notices
02.04.2013	0.9	<ul style="list-style-type: none"><li>▪ Term TLS mistyping fixed.</li><li>▪ Improved chapter <u>1.3 Identifying the card application</u>.</li><li>▪ Improved description for <u>MANAGE SECURITY ENVIRONMENT</u> command.</li><li>▪ Fixed chapters <u>3.1</u>, <u>3.2.1</u>, <u>3.2.2</u> and <u>3.3</u> by according to previous mentioned improvement.</li></ul>
15.05.2013	0.91	<ul style="list-style-type: none"><li>▪ Clarifications for protocol T0.</li><li>▪ Clarifications for application identification.</li></ul>
21.05.2013	1.0	Specification release
10.06.2013	1.05	<ul style="list-style-type: none"><li>▪ Document reference links fixed</li><li>▪ Wording improvements</li></ul>
29.09.2013	1.10	<ul style="list-style-type: none"><li>▪ Clarification about ATR historical bytes</li></ul>
11.03.2014	1.20	<ul style="list-style-type: none"><li>▪ Added introductory section " Differentiation to previous versions of EstEID card application</li></ul>
14.03.2017	1.30	<ul style="list-style-type: none"><li>• Fixed MSB to LSB in chapter 2.5.1</li><li>• Introduced FCI and added chapter 1.4 File Control Information (FCI) on EstEID application</li><li>• Changed SELECT FILE command P2 to use 0C<sub>hex</sub> instead of 04<sub>hex</sub> all over the document. This is strongly recommended P2 value to be used for SELECT FILE command for those who do not know what is its purpose!</li></ul>



## Introduction

The aim of this document is to specify the functions, the data content and the interface of the security chip along with the smart card application of the Estonian national public key infrastructure (EstEID).

The reader of this document is expected to be familiar with smart card and chip application related topics. Also the reader should have a fluent knowledge of the Estonian PKI.

## Prerequisites to the Smart Card

The Estonian Electronic ID-card application is designed to run on an integrated circuit, a chip with Java Global Platform operating system and a contact based interface. It is assumed that the chip provides EEPROM capacity of 80 kilobytes or more.

The operating system and the chip shall be certified at least EAL4+ against Common Criteria protection profiles 002 and 035.

## Document Scope

This specification describes the intended function and behaviour of the application being the default selectable one on the chip. The scope of this specification is strictly limited to the use within the frame of the Estonian PKI.

The reaction and behaviour of the application and the card to experiments, tests and attack attempts is out of scope of this document.

In this document the following topics are covered:

- Usage;
- Personalisation;
- Configuration; and
- Maintenance.

The usage of the chip application shall be fully covered by this document. The personalisation, configuration and maintenance can involve the specification of the chip OS, of the chip hardware and also the PKI policy.



## Differentiation to previous versions of EstEID card application

ID	previous versions	version 3.5.1. - 3.5.3	version 3.5.7	comments
drop i	CMK3	-		Java card has its own card manager. No need to misuse the EstEID application to manage the potential loading of applications.
drop ii	RootCA certificate	-		RootCA certificate is always checked online. There's no need to keep a place for offline checks.
drop iii	Passphrase (DESkeys)	-		Only PIN shall be used for authentication, signing and decryption function.
drop iv	Decryption function for signature keypair	-		The signature key and certificate shall be exclusively used for qualified digital signatures.
add α	-	Introduction of AID		An application identifier tag has been introduced.
add β	-	Extension of version ID		The version ID value has been extended from 2 to 3 bytes.
add γ	-	Option to support SHA-2		Beside SHA-1 also SHA-2 support has been added.
add δ	-	Enhanced management of life cycles		Running on a java card the life cycle of the card and the applet (the EstEID application) are clearly to be distinguished and managed separately.
add 7	-	-	Introduction of FCI	The file control information (FCI) has been activated. It delivers the AID.

## Document legend



Tips



Notes



Important information or warnings

### Operators

**||** – Operator marking the concatenation operation.

**+** – Operator marking the adding operation.

**HEX** – Marks that the number is presented in hexadecimal format.

**DEC** – Marks that the number is presented in decimal format.

**BIT** – Marks that the number is presented in binary format.

**0x** – Marks that the following number is presented in hexadecimal format.



## Table of contents

<b>1.</b>	<b>Chip and card application .....</b>	<b>8</b>
1.1.	Answer to reset .....	8
1.2.	Card application .....	8
1.3.	Identifying the card application .....	8
1.4.	File Control Information (FCI) on EstEID application .....	9
1.5.	Card application file system structure .....	10
1.6.	Objects in the card application .....	11
1.7.	Card application principles .....	11
<b>2.</b>	<b>Card application objects, their details and general operations .....</b>	<b>12</b>
2.1.	Personal data file .....	12
2.1.1.	Reading contents of Personal Data file .....	13
2.2.	PIN1, PIN2 and PUK code .....	14
2.2.1.	Verify PIN1, PIN2 or PUK code .....	15
2.2.2.	Changing PIN1, PIN2 or PUK code .....	16
2.2.3.	Unblocking PIN1 or PIN2 code .....	17
2.2.4.	Reading PIN1, PIN2, or PUK code counter .....	18
2.3.	Certificates .....	20
2.3.1.	Reading certificate files .....	20
2.4.	Cardholder secret keys .....	22
2.4.1.	Reading public key of cardholder secret key .....	24
2.4.2.	Reading secret key information .....	24
2.4.3.	Reading key references for active keys .....	25
2.5.	Card application management keys: CMK_PIN, CMK_CERT & CMK_KEY .....	26
2.5.1.	Deriving card application management keys .....	27
2.6.	Miscellaneous information .....	27
2.6.1.	Reading EstEID application version .....	28
2.6.2.	Reading CPLC data .....	28
2.6.3.	Reading data for available memory on chip .....	29
<b>3.</b>	<b>Card application general operations .....</b>	<b>30</b>
3.1.	Calculating the response for TLS challenge .....	30
3.2.	Calculating the electronic signature .....	31



---

3.2.1.	Calculating the electronic signature with providing pre-calculated hash .....	32
3.2.2.	Calculating the electronic signature with internal hash calculating ...	33
3.3.	<b>Decrypting public key encrypted data .....</b>	<b>35</b>
<b>4.</b>	<b><u>Card application managing operations.....</u></b>	<b><u>37</u></b>
4.1.	<b>Secure channel communication .....</b>	<b>37</b>
4.1.1.	Mutual Authentication .....	38
4.1.2.	Channel securing .....	39
4.2.	<b>PIN1, PIN2 and PUK replacement .....</b>	<b>41</b>
4.3.	<b>Certificate replacement.....</b>	<b>42</b>
4.4.	<b>New RSA key pair generation .....</b>	<b>42</b>
<b>5.</b>	<b><u>Card application security structure.....</u></b>	<b><u>44</u></b>
<b>6.</b>	<b><u>Card application constants .....</u></b>	<b><u>45</u></b>
<b>7.</b>	<b><u>APDU protocol.....</u></b>	<b><u>45</u></b>
7.1.	<b>Card possible response in case of protocol T0 .....</b>	<b>47</b>
7.2.	<b>Command APDU .....</b>	<b>48</b>
7.2.1.	SELECT FILE.....	49
7.2.2.	READ RECORD.....	50
7.2.3.	READ BINARY .....	51
7.2.4.	GET RESPONSE .....	51
7.2.5.	GET DATA.....	52
7.2.6.	GET CHALLENGE .....	53
7.2.7.	VERIFY .....	53
7.2.8.	CHANGE REFERENCE DATA.....	54
7.2.9.	RESET RETRY COUNTER .....	55
7.2.10.	MANAGE SECURITY ENVIRONMENT.....	56
7.2.11.	INTERNAL AUTHENTICATE .....	56
7.2.12.	MUTUAL AUTHENTICATE .....	57
7.2.13.	PERFORM SECURITY OPERATION.....	57
7.2.13.1.	HASH .....	58
7.2.13.2.	DECIPHER .....	58
7.2.13.3.	COMPUTE DIGITAL SIGNATURE.....	59
7.2.14.	REPLACE PINS (SECURE).....	59
7.2.15.	GENERATE KEY (SECURE) .....	60
7.2.16.	REPLACE CERTIFICATE (SECURE) .....	61



---

<b>7.3.</b>	<b>Error response APDU messages.....</b>	<b>61</b>
<b>7.4.</b>	<b>Message chaining.....</b>	<b>62</b>
<b>7.5.</b>	<b>Extended APDU.....</b>	<b>63</b>
<b>Abbreviations.....</b>		<b>64</b>
<b>Terms.....</b>		<b>65</b>
<b>References.....</b>		<b>65</b>
<b>List of Tables.....</b>		<b>66</b>
<b>List of Pictures.....</b>		<b>66</b>
<b>APPENDIX .....</b>		<b>68</b>
<b>Reset the chip with EstEID card application installed on .....</b>		<b>68</b>
<b>PIN1, PIN2 and PUK operations.....</b>		<b>68</b>
<b>Navigate to DF FID EEEE<sub>hex</sub> .....</b>		<b>69</b>
<b>Select EF FID 5044<sub>hex</sub> and read all of its contents .....</b>		<b>69</b>
<b>Read certificate files .....</b>		<b>70</b>
Read authentication certificate using multiple C-APDUs.....		70
Read signature certificate using extended C-APDU .....		71
<b>Read cardholder secret keys info from file 0013<sub>hex</sub>.....</b>		<b>72</b>
<b>Read miscellaneous information .....</b>		<b>72</b>
<b>Card application general operations.....</b>		<b>73</b>
Calculate response for TLS challenge .....		73
Calculate electronic signature from pre-calculated SHA1 hash.....		73
Perform deciphering operation.....		73
<b>Card application managing operations.....</b>		<b>74</b>
Replace cardholder PINs/PUK codes .....		74
Generate new key pair.....		75
Replace Certificates .....		78
Replacing of Authentication Certificate .....		78
Replacing of Signing Certificate .....		87



## 1. Chip and card application

### 1.1. Answer to reset

Every contact card respond to reset with sequence of bytes called Answer To Reset<sup>1</sup>(ATR).

The ATR gives information about the electrical communication protocol and the chip itself. It is mainly linked with the underlying integrated circuit (chip) and also the Java operating system on it. There is a block of historical bytes that can be used to indicate the purpose of the chip card.

The ATR can be different depending on if the reset is the first since power-up (Cold ATR) or not (Warm ATR). The meaningful info of ATR can be read from historical bytes of Cold ATR. In ANSI encoding they can be represented as “eID / PKI”. Together with a category indicator byte the historical bytes form a string of 10 bytes with the value "FE 65 49 44 20 2F 20 50 4B 49<sub>hex</sub>".

The ATR can also be different when the chip, that carries the EstEID chip application, gets replaced.



The ATR cannot be used to identify the EstEID chip card application. The identification procedure is described in chapter [1.3 Identifying the card application](#).

### 1.2. Card application

EstEID card application is implemented on top of the JavaCard framework version 2.2.2. It enables operations which are required for PKI procedures. Card application has 3 different internal lifecycle states:

- Blank – Clean card application
- Personalised – Card application with cardholder personal information, PINs and CMKs.
- Live – Card application also has PKI objects.

Current specification concerns only the Live lifecycle state of card application.

### 1.3. Identifying the card application

EstEID application is the default selected application on the card. The card application can only be identified by selecting the card application with its AID and then identifying the card application version number from the card. The procedure for identifying the application version number is described in chapter [2.6.1 Reading EstEID application version](#). The card should return version 3.5.7 or higher.

Card application identification should be performed with following operations:

---

<sup>1</sup> See ISO 7816-3: Identification cards — Integrated circuit cards — Part 3: Cards with contacts — Electrical interface and transmission protocols



- 1) Application selection should be performed by executing following SELECT FILE command:

CLA	INS	P1	P2	Lc	Data (AID)
00 <sub>hex</sub>	A4 <sub>hex</sub>	04 <sub>hex</sub>	0C <sub>hex</sub>	0F <sub>hex</sub>	D23300000045737445494420763335 <sub>hex</sub>

The card should respond with status word 9000<sub>hex</sub>.



The application that gets selected is an application that is already selected by default on the card. Previous command is only a procedure for double checking that correct application is selected.

- 2) Card application version identification should be performed as defined in chapter 2.6.1 Reading EstEID application version.



A proper and biunique identification of the card requires reading both the AID and also the version ID. The AID specifies the application and its features. The third digit of the version ID defines the application build.

#### 1.4. File Control Information (FCI) on EstEID application

The EstEID application is set to be the default selectable application on the chip. The application supports the FCI. It is provided as immediate answer to the Select Master File command (0x00A40000).

Application specific tags are 0x4F and 0xDE.

The tag 0x4F, application identifier, is present in all EstEID applications as from version 3.6 and higher. It contains the AID value.

The tag 0xDE, development version identifier, is only present in development versions of the EstEID application. Release versions do not have this tag. The tag is meant to contain information that identifies the development version built and loaded onto a card: Initials of the developer (2 bytes or more), the codebase ID (4 bytes) and the date of the build (6 bytes, YYYYMMDD).

*FCI Example – EstEID chip application release version:*

6F 27 62 25 82 01 38 83 02 3F 00 84 02 4D 46 85 02 7F FF 4F 0F D2 33 00 00 00 45 73  
74 45 49 44 20 76 33 35 8A 01 05 64 00

4F = Application Identifier (AID)

0F = Length in bytes (15 bytes dec)

D2 33 00 00 00 45 73 74 45 49 44 20 76 33 36 = EE EstEID v36 (ISO8859-1)

*FCI Example – EstEID chip application development version:*

6F 39 62 35 82 01 38 83 02 3F 00 84 02 4D 46 85 02 7F FF 4F 0F D2 33 00 00 00 45 73  
74 45 49 44 20 76 33 35 DE 10 52 57 06 5A 40 93 20 15 11 20 00 00 00 00 00 00 8A 01  
05 64 00

DE = Development version identifier (DVI)  
10 = Length in bytes (16 bytes dec)  
52 57 = Developer initials, RW (ISO8859-1)  
06 5A 40 93 = Codebase ID  
20 15 11 20 = Date (YYYYMMDD)  
00 00 00 00 00 00 = Padding (RFU)

## 1.5. Card application file system structure

EstEID application derives file system attributes and functionalities from ISO 7816-4. The structure of the file system can be seen on the figure on the right.

Card application specific information is held in DF FID EEEE<sub>hex</sub>. The heart data of card application can be considered the files which hold certificates (EF FID AACE<sub>hex</sub> & DDCE<sub>hex</sub>) and card holder personal data (EF FID 5044<sub>hex</sub>). The meaning of other files is to hold information about card application internal counters and references.

The reading operation, of files seen on the figure on the right, is specified in subchapters of chapter 2 Card application objects, their details and general operations.

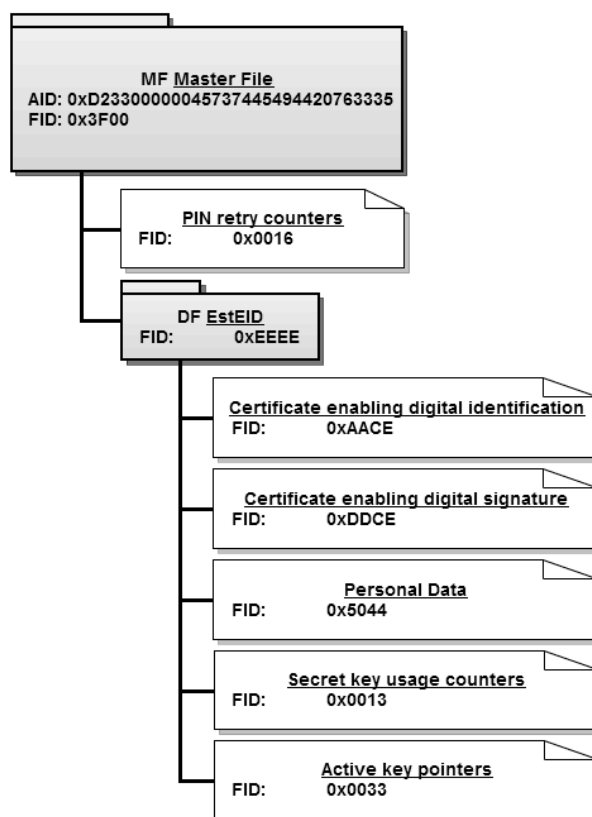


Figure 1-1 EstEID filesystem diagram



## 1.6. Objects in the card application

Table 1-1 The functions of EstEID security chip objects in the card application	
Object	Function description
PIN1	The authorisation of the cardholder: 1) for getting access to the authentication key procedures. 2) for the execution of the following operations: a) the generation of new key pairs b) the loading of certificates
PIN2	For getting access to signature key.
PUK	The unblocking of PIN codes when they have been blocked after number of allowed consecutive incorrect entries.
Authentication certificate	The certificate for cardholder identification.
Signature certificate	The certificate calculating and checking the cardholder's electronic signature.
Authentication key pair (x2)	<ul style="list-style-type: none"><li>▪ Key pair that is actively used for cardholder authentication procedures.</li><li>▪ Optional idle key pair for a potential future replacement of active authentication key pair.</li></ul>
Signature key pair (x2)	<ul style="list-style-type: none"><li>▪ Key pair that is actively used for digital signing procedures.</li><li>▪ Optional idle key pair for a potential future replacement of active signature key pair.</li></ul>
Cardholder personal data file	Includes the cardholder's personal data.
CMK_PIN	3DES key which is used to secure the PIN code replacement procedure.
CMK_KEY	3DES key which is used to authorise the new key pair generation.
CMK_CERT	3DES key which is used to form the secure command series to load the user certificates.

## 1.7. Card application principles

Card application is realised on Java chip platform. It implements functionalities derived from ISO 7816-4 and ISO 7816-8 as much as it is required for electronic identification and signing operations. Therefore many of the functionalities we know from ISO 7816 are not implemented. This kind of minimalistic implementation should make chip using easier and clearer for software developers.

Card application supports both T0 and T1 transport protocol.

DES cryptographic operations are performed using ISO 9797-1 padding method 2.

Card application uses for PKI operations 2048 bit RSA with method PKCS#1 version 1.5.

Card application has three different use cases for different card application interfaces, which are inside the card implemented as environments:

- Public environment – Reading card application data objects and changing PIN/PUK codes.
- PKI environment – For performing PKI operations in card application.
- Card application authority environment – Replacing PIN/PUK and certificate objects. Generating new key pairs for cardholder.

All operations with RSA key pairs are authorised by verifying the cardholder with PIN1 or PIN2.

## 2. Card application objects, their details and general operations

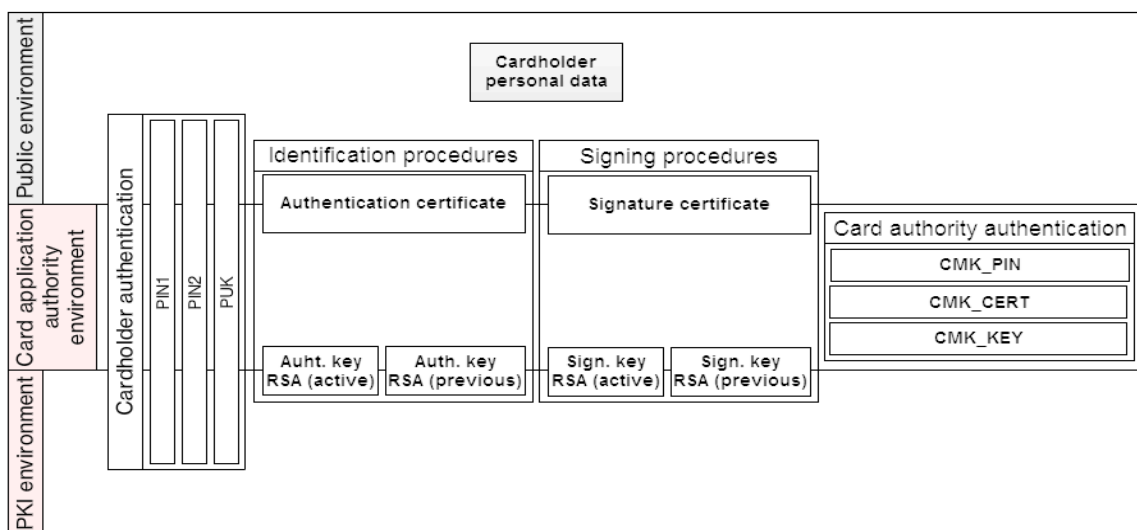


Figure 2-1 Card application objects

Card application contains objects which are used for different procedures. These procedures can be for public, PKI or card application authority environment use. Figure above shows in which environment objects are used. Following sub-chapters describe the objects and their operations.

### 2.1. Personal data file

EstEID card has on the card personal information of the cardholder. The same information is also included digitally on the card application (except photo and signature images). Cardholder personal data is held in file with File Identifier (FID) 5044<sub>hex</sub> or 'PD' as ASCII.

The cardholder personal data file includes the records given below. It is a variable-length formatted file where every record in it can differ in length. Records which have marked with maximum length can contain data with various lengths. All the data in these records are coded according to ANSI code page 1252. The records that have the maximum length marked in [Table 2-1 Personal Data file contents](#), could have various length of data up to maximum marked. If given records do not contain meaningful data they will be filled with a placeholder, one space character (20<sub>hex</sub> as ANSI).



Table 2-1 Personal Data file contents		
Record no.	Content	Length or maximum length
1	Surname	max 28 <sub>dec</sub> bytes
2	First name line 1	max 15 <sub>dec</sub> bytes
3	First name line 2	max 15 <sub>dec</sub> bytes
4	Sex Values: 'M' – Male 'N' – Female	1 <sub>dec</sub> bytes
5	Nationality (3 letters) According to ISO 3166-1 alpha-3.	3 <sub>dec</sub> bytes
6	Birth date (dd.mm.yyyy)	10 <sub>dec</sub> bytes
7	Personal identification code	11 <sub>dec</sub> bytes
8	Document number	9 <sub>dec</sub> bytes
9	Expiry date (dd.mm.yyyy)	10 <sub>dec</sub> bytes
10	Place of birth	max 35 <sub>dec</sub> bytes
11	Date of issuance (dd.mm.yyyy)	10 <sub>dec</sub> bytes
12	Type of residence permit	max 50 <sub>dec</sub> bytes
13	Notes line 1	max 50 <sub>dec</sub> bytes
14	Notes line 2	max 50 <sub>dec</sub> bytes
15	Notes line 3	max 50 <sub>dec</sub> bytes
16	Notes line 4	max 50 <sub>dec</sub> bytes

### 2.1.1. Reading contents of Personal Data file

To read the personal information of the cardholder following operations must be performed:

- 1) Always is good to select the root file of the card application called Master File (MF) with command SELECT FILE. This is only if you do not know currently selected file on the card application. After card reset or insertion to card reader this file is initially selected and has no actual need to perform this operation.

CLA	INS	P1	P2	Le
00 <sub>hex</sub>	A4 <sub>hex</sub>	00 <sub>hex</sub>	0C <sub>hex</sub>	00 <sub>hex</sub>

- 2) Select directory file EEEE<sub>hex</sub> by using command SELECT FILE again.

CLA	INS	P1	P2	Lc	Data (FID)
00 <sub>hex</sub>	A4 <sub>hex</sub>	01 <sub>hex</sub>	0C <sub>hex</sub>	02 <sub>hex</sub>	EEEE <sub>hex</sub>

- 3) Select Personal Data file with FID 5044<sub>hex</sub> by using command SELECT FILE once again.

CLA	INS	P1	P2	Lc	Data (FID)
00 <sub>hex</sub>	A4 <sub>hex</sub>	02 <sub>hex</sub>	0C <sub>hex</sub>	02 <sub>hex</sub>	5044 <sub>hex</sub>

After given command the Personal Data file is selected and it is possible to read the contents of the file.

- 4) Records that the Personal Data file contains are specified in table [Personal Data file contents](#). For this example let's read record no. 7 „Personal identification code“, with command [READ RECORD](#).



For current example let the record no. 7 value be '47101010033'.

CLA	INS	P1 (record no)	P2	Le
00 <sub>hex</sub>	B2 <sub>hex</sub>	07 <sub>hex</sub>	04 <sub>hex</sub>	00 <sub>hex</sub>

In case Le is set to 00<sub>hex</sub> the length of the record is not known and as a result card application will respond with the data of the record in R-APDU. The Le field can as well have the exact length value for record or value larger than record. For the last case the data from the next record(s) will be also returned in R-APDU.

For successful read card application responds with R-APDU:

Data	SW1	SW2
3437313031303130303333 <sub>hex</sub>	90 <sub>hex</sub>	00 <sub>hex</sub>



It must be considered that if protocol T0 is used and Le has value 00<sub>hex</sub> or it is absent, then the card will respond with status 61XX<sub>hex</sub>. How to operate in given situation is described in chapter [7.1 Card possible response in case of protocol T0](#).

## 2.2. PIN1, PIN2 and PUK code

Verification by card application can be done with 3 different methods. These methods are called PIN1, PIN2 and PUK code. All these methods have their own card application operational purpose. Details about these verification methods are specified in the table below.

Table 2-2 PIN1, PIN2 and PUK				
Method	Length in bytes			Definition
	Min	Max	Initial	
PIN1	4 <sub>dec</sub>	12 <sub>dec</sub>	4 <sub>dec</sub>	For operations with Authentication key.
PIN2	5 <sub>dec</sub>	12 <sub>dec</sub>	5 <sub>dec</sub>	For operations with Digital Signature key.
PUK	8 <sub>dec</sub>	12 <sub>dec</sub>	8 <sub>dec</sub>	For unblocking PIN1 or PIN2 codes.

Card application does not allow using PIN and PUK codes with lengths which are outside of the limits range. Initial lengths of the application are given in the table above. The length of PIN and PUK codes may vary if cardholder has changed the default value. The host applications communicating with the card application must support all lengths of PIN and PUK codes marked in the table above.

All verification codes consist of ASCII digits from '0' to '9'. As well, all codes must be transmitted to card application in ASCII format.



Card application supports as well all other ASCII characters for verification. However other characters than digits are not supported by the use cases.

PIN and PUK codes cannot be read from the card application. These codes can only be verified by the card application.

The cardholder authentication with PIN1, PIN2 or PUK is conducted by the command VERIFY. The given operation may be executed without restrictions.

All codes have separate retry counter which are initially and reset to value 3 after successful verification. Every unsuccessful verification results in a decrease of the corresponding code's retry counter. If retry counter has reached to 0, given verification method is blocked. Blocked PIN codes can be unblocked using RESET\_RETRY\_COUNTER command. However if PUK code retry counter reaches to 0, card application PIN and PUK codes can be unblocked and changed only by the card application authority.

### 2.2.1. Verify PIN1, PIN2 or PUK code

Cardholder authentication is done by verifying PIN and PUK codes. After successful verification procedure application general operations will be accessible.



It is strongly advised to perform cardholder authentication with PIN or PUK verification on external pin pad device.



Keep in mind that unsuccessful operation of command VERIFY decreases pin retry counter value by one.



For PIN and PUK examples below let's use following PIN and PUK code values:

- PIN1 – '1234' as ASCII or 31323334<sub>hex</sub>
- PIN2 – '12345' as ASCII or 3132333435<sub>hex</sub>
- PUK – '12345678' as ASCII or 3132333435363738<sub>hex</sub>

To verify PIN1 it is needed to execute the following command VERIFY:

CLA	INS	P1	P2	Lc	Data (PIN1 as ASCII)
00 <sub>hex</sub>	20 <sub>hex</sub>	00 <sub>hex</sub>	01 <sub>hex</sub>	04 <sub>hex</sub>	31323334 <sub>hex</sub>

To verify PIN2 it is needed to execute the following command VERIFY:

CLA	INS	P1	P2	Lc	Data (PIN2 as ASCII)
00 <sub>hex</sub>	20 <sub>hex</sub>	00 <sub>hex</sub>	02 <sub>hex</sub>	05 <sub>hex</sub>	3132333435 <sub>hex</sub>

To verify PUK it is needed to execute the following command VERIFY:

CLA	INS	P1	P2	Lc	Data (PUK as ASCII)
00 <sub>hex</sub>	20 <sub>hex</sub>	00 <sub>hex</sub>	00 <sub>hex</sub>	08 <sub>hex</sub>	3132333435363738 <sub>hex</sub>

For successful operation card application will respond with status 9000<sub>hex</sub> and verification in the application is flagged as verified. For other possible R-APDU statuses look chapter 7.2.7 VERIFY.

### 2.2.2. Changing PIN1, PIN2 or PUK code

The values of PIN1, PIN2 and PUK codes can be replaced with the command CHANGE REFERENCE DATA if given code is not blocked.

The values of PIN1 and PIN2 codes can be replaced also with PUK verification using command RESET RETRY COUNTER. For examples for given command see chapter 2.2.3 Unblocking PIN1 or PIN2 code.



The values of previous PIN or PUK code and new PIN or PUK code have to be different when changing the code's value.

As a result for successful operation of commands above the code will be replaced and retry counter reset for given operable code.



For given example let the PIN and PUK codes have the same values as in chapter 2.2.1 Verify PIN1, PIN2 or PUK code.

To replace PIN1 code with '54321' it is needed to executed the following command CHANGE REFERENCE DATA:

CLA	INS	P1	P2	Lc	Data
00 <sub>hex</sub>	24 <sub>hex</sub>	00 <sub>hex</sub>	01 <sub>hex</sub>	09 <sub>hex</sub>	313233343534333231 <sub>hex</sub>

To replace PIN2 code with '654321' it is needed to executed the following command CHANGE REFERENCE DATA:

CLA	INS	P1	P2	Lc	Data
00 <sub>hex</sub>	24 <sub>hex</sub>	00 <sub>hex</sub>	02 <sub>hex</sub>	0B <sub>hex</sub>	3132333435363534333231 <sub>hex</sub>

To replace PUK code with '987654321' it is needed to executed the following command CHANGE REFERENCE DATA:

CLA	INS	P1	P2	Lc	Data
00 <sub>hex</sub>	24 <sub>hex</sub>	00 <sub>hex</sub>	00 <sub>hex</sub>	11 <sub>hex</sub>	3132333435363738393837363534333231 <sub>hex</sub>

For successful operation card application will respond with status 9000<sub>hex</sub>. For other possible R-APDU statuses look chapter 7.2.8 CHANGE REFERENCE DATA.



### 2.2.3. Unblocking PIN1 or PIN2 code

When PIN codes retry counter values has decremented to value 0 and get blocked, it is possible to unblock them by resetting the retry counter. This operation is possible with command RESET RETRY COUNTER.



For given example let the PUK code have the same values as in chapter 2.2.1 Verify PIN1, PIN2 or PUK code.

1) To unblock PIN codes with PUK code verification it is needed to do following:  
First it is needed to verify the PUK code, with executing the following command VERIFY:

CLA	INS	P1	P2	Lc	Data (PUK as ASCII)
00 <sub>hex</sub>	20 <sub>hex</sub>	00 <sub>hex</sub>	00 <sub>hex</sub>	08 <sub>hex</sub>	3132333435363738 <sub>hex</sub>

After given command is executed and successful response returned it is possible to reset retry counter of PIN codes.



PIN1 and PIN2 can only be unblocked if they are in blocked state.

a) To unblock and reset retry counter of PIN1 with pre-verified PUK it is needed to executed the following command RESET RETRY COUNTER:

CLA	INS	P1	P2	Le
00 <sub>hex</sub>	2C <sub>hex</sub>	03 <sub>hex</sub>	01 <sub>hex</sub>	00 <sub>hex</sub>

b) Or to unblock and reset retry counter of PIN2 with pre-verified PUK it is needed to executed the following command RESET RETRY COUNTER:

CLA	INS	P1	P2	Le
00 <sub>hex</sub>	2C <sub>hex</sub>	03 <sub>hex</sub>	02 <sub>hex</sub>	00 <sub>hex</sub>

After successful execution of previous commands PIN codes get unblocked and retry counters reset.

2) To unblock by changing the PIN codes with PUK code verification it is needed to do following:



Keep in mind that unsuccessful operation of following command RESET RETRY COUNTER decreases PUK retry counter value by one.



The values of previous internal PIN code and new PIN code can have the same values.



- a) To unblock and reset retry counter of PIN1 with pre-verified PUK it is needed to executed the following command RESET RETRY COUNTER:

CLA	INS	P1	P2	Lc	Data (PUK    new PIN1 '4321')
00 <sub>hex</sub>	2C <sub>hex</sub>	00 <sub>hex</sub>	01 <sub>hex</sub>	0C <sub>hex</sub>	3132333435363738 <sub>hex</sub>    34333221 <sub>hex</sub>

- b) Or to unblock and reset retry counter of PIN2 with pre-verified PUK it is needed to executed the following command RESET RETRY COUNTER:

CLA	INS	P1	P2	Lc	Data (PUK    new PIN2 '54321')
00 <sub>hex</sub>	2C <sub>hex</sub>	00 <sub>hex</sub>	02 <sub>hex</sub>	0D <sub>hex</sub>	3132333435363738 <sub>hex</sub>    3534333221 <sub>hex</sub>

#### 2.2.4. Reading PIN1, PIN2, or PUK code counter

The current retry counter values of PIN and PUK codes can be read from the card application from PIN retry counters file with FID 0016<sub>hex</sub>. It is variable-length formatted file containing 3 records in following corresponding sequence for PIN1, PIN2 and PUK code.

Table 2-3 EF FID 0016 <sub>hex</sub> contents		
Record no.	Verification method	Description
1	PIN1	TLV data containing maximum and current retry counter value and unblock reference.
2	PIN2	TLV data containing maximum and current retry counter value and unblock reference.
3	PUK	TLV data containing maximum and current retry counter value.

Before the PIN retry counters file can be read it must be selected:

- a) First it must be sure that the MF is selected. Execute MF selection with command SELECT FILE:

CLA	INS	P1	P2	Le
00 <sub>hex</sub>	A4 <sub>hex</sub>	00 <sub>hex</sub>	0C <sub>hex</sub>	00 <sub>hex</sub>

- b) Secondly execute selection for EF FID 0016<sub>hex</sub> with command SELECT FILE:

CLA	INS	P1	P2	Lc	Data
00 <sub>hex</sub>	A4 <sub>hex</sub>	02 <sub>hex</sub>	0C <sub>hex</sub>	02 <sub>hex</sub>	0016 <sub>hex</sub>

When the EF FID 0016 is selected there can be performed reading operations with command READ RECORD to get the PIN and PUK codes counters:



- a) To read retry counter of PIN1 it is needed to executed the following command  
**READ RECORD:**

CLA	INS	P1 (record no)	P2	Le
00 <sub>hex</sub>	B2 <sub>hex</sub>	01 <sub>hex</sub>	04 <sub>hex</sub>	00 <sub>hex</sub>

For successful reading operation chip responds with following R-APDU where X marks remaining tries for PIN1:

Data						SW1	SW2
Max. tries		Remaining tries		Unblock reference			
Tag    Length	Value	Tag    Length	Value	Tag    Length	Value		
80 <sub>hex</sub>    01 <sub>hex</sub>	03 <sub>hex</sub>	90 <sub>hex</sub>    01 <sub>hex</sub>	0X <sub>hex</sub>	83 <sub>hex</sub>    02 <sub>hex</sub>	0000 <sub>hex</sub>	90 <sub>hex</sub>	00 <sub>hex</sub>

- b) To read retry counter of PIN2 it is needed to executed the following command  
**READ RECORD:**

CLA	INS	P1 (record no)	P2	Le
00 <sub>hex</sub>	B2 <sub>hex</sub>	02 <sub>hex</sub>	04 <sub>hex</sub>	00 <sub>hex</sub>

For successful reading operation chip responds with following R-APDU where X marks remaining tries for PIN2:

Data						SW1	SW2
Max. tries		Remaining tries		Unblock reference			
Tag    Length	Value	Tag    Length	Value	Tag    Length	Value		
80 <sub>hex</sub>    01 <sub>hex</sub>	03 <sub>hex</sub>	90 <sub>hex</sub>    01 <sub>hex</sub>	0X <sub>hex</sub>	83 <sub>hex</sub>    02 <sub>hex</sub>	0000 <sub>hex</sub>	90 <sub>hex</sub>	00 <sub>hex</sub>

- c) To read retry counter of PUK it is needed to executed the following command  
**READ RECORD:**

CLA	INS	P1 (record no)	P2	Le
00 <sub>hex</sub>	B2 <sub>hex</sub>	03 <sub>hex</sub>	04 <sub>hex</sub>	00 <sub>hex</sub>

For successful reading operation chip responds with following R-APDU where X marks remaining tries for PUK:

Data					SW1	SW2
Max. tries		Remaining tries				
Tag    Length	Value	Tag    Length	Value			
80 <sub>hex</sub>    01 <sub>hex</sub>	03 <sub>hex</sub>	90 <sub>hex</sub>    01 <sub>hex</sub>	0X <sub>hex</sub>		90 <sub>hex</sub>	00 <sub>hex</sub>



## 2.3. Certificates

EstEID application contains 2 certificates in the card:

- Certificate for cardholder authentication operations.
- Certificate for cardholder digital signing operations.

Certificate files in the card application are located in DF with FID  $EEEE_{hex}$ . Authentication certificate file has FID  $AACE_{hex}$  and digital signing certificate file has FID  $DDCE_{hex}$ . Certificate files are transparent files and can be read with command READ BINARY. In the card application certificate files have fixed length memory allocated as there may be a need to write new certificates which have slightly different length. Certificate files are formatted as ASN.1 DER.

To fit the certificate into the file, certificate data is padded according to ISO 9797-1 padding method 2. The padding must be appended to a whole number of bytes long data:

600 <sub>hex</sub> bytes	
Certificate bytes	Padding start indicator    Padding zero bytes until the end of file
3082 <sub>hex</sub> ... XX <sub>hex</sub>	80 <sub>hex</sub>    00 <sub>hex</sub> ... 00 <sub>hex</sub>

Padding example for certificate with length 5F9<sub>hex</sub> bytes:

Certificate data (5F9 <sub>hex</sub> bytes)	Padding (7 <sub>dec</sub> bytes)
3082 <sub>hex</sub> ... XX <sub>hex</sub>	8000000000000000 <sub>hex</sub>

### 2.3.1. Reading certificate files

Both certificate files from the card application can be read in the similar way. There are 2 ways to read the entire certificate file from the card. The reading of the file can be done by using command READ BINARY. The easiest way to read the file is to use extended length APDU for it. Another way is to use sequence of READ BINARY command by increasing the file reading offset for every next command until the whole file data is returned.

Certificate files are located in DF  $EEEE_{hex}$ . First navigate to given directory by:

- a) selecting the application MF with command SELECT FILE:

CLA	INS	P1	P2	Le
00 <sub>hex</sub>	A4 <sub>hex</sub>	00 <sub>hex</sub>	0C <sub>hex</sub>	00 <sub>hex</sub>

- b) and selecting directory file  $EEEE_{hex}$  by using command SELECT FILE again.

CLA	INS	P1	P2	Lc	Data (FID)
00 <sub>hex</sub>	A4 <sub>hex</sub>	01 <sub>hex</sub>	0C <sub>hex</sub>	02 <sub>hex</sub>	$EEEE_{hex}$

Next step is to select desired certificate file for reading by using command SELECT FILE once again:

- a) For selecting Authentication certificate file use:



CLA	INS	P1	P2	Lc	Data (FID)
00 <sub>hex</sub>	A4 <sub>hex</sub>	02 <sub>hex</sub>	0C <sub>hex</sub>	02 <sub>hex</sub>	AACE <sub>hex</sub>

a) For selecting Digital Signature certificate file use:

CLA	INS	P1	P2	Lc	Data (FID)
00 <sub>hex</sub>	A4 <sub>hex</sub>	02 <sub>hex</sub>	0C <sub>hex</sub>	02 <sub>hex</sub>	DDCE <sub>hex</sub>

Now certificate file is ready for reading operations:



This are just examples showing one possible way of doing it. Please refer to ISO7816-4 to exploit other methods and solutions.

- 1) First method for reading the file is using extended APDU. For this it is needed to send command READ BINARY in following format:

CLA	INS	P1	P2	Ext. indicator	APDU	Le (length of certificate file)
00 <sub>hex</sub>	B0 <sub>hex</sub>	00 <sub>hex</sub>	00 <sub>hex</sub>	00 <sub>hex</sub>		0600 <sub>hex</sub>

- 2) Second method for reading the file is by sending multiple READ BINARY commands by changing the file reading offset for every next command until the whole file has been read. Data of the result has to be concatenated together.

Keep in mind that certificate file has length of 600<sub>hex</sub> bytes but certificate file data is actually shorter in length. The actual length of the certificate can be derived from the first response of the READ BINARY command because certificate data is in ASN.1 DER encoded format. 3rd and 4th byte of the 1st response hold the actual length of certificate data.

Actual length of the file can be calculated as  $4_{\text{dec}} + (3\text{rd byte} \parallel 4\text{th byte})_{\text{hex}}$  from the response bytes.

a) First READ BINARY command in sequence:

CLA	INS	P1	P2	Le
00 <sub>hex</sub>	B0 <sub>hex</sub>	00 <sub>hex</sub>	00 <sub>hex</sub>	00 <sub>hex</sub> (actually expects 256 <sub>dec</sub> )

Command response:

Data				SW1	SW2
Tag	Length	Value			
30 <sub>hex</sub>	82 <sub>hex</sub>    04A5 <sub>hex</sub>	252 <sub>dec</sub> bytes of data		90 <sub>hex</sub>	00 <sub>hex</sub>

3rd and 4th bytes in 1st response have values 04<sub>hex</sub> and A5<sub>hex</sub>. From this we can calculate the length of certificate data:

$$4_{\text{dec}} + (04_{\text{hex}} \parallel A5_{\text{hex}}) = 04A9_{\text{hex}}$$



256<sub>dec</sub> bytes have already been read. There are 03A9<sub>hex</sub> bytes still waiting to be read.

- b) 2nd, 3rd and 4th time read another 256<sub>dec</sub> bytes from the chip with READ BINARY command by increasing file reading offset every time with 256<sub>dec</sub>:

CLA	INS	P1	P2	Le		
00 <sub>hex</sub>	B0 <sub>hex</sub>	01 <sub>hex</sub>	00 <sub>hex</sub>	00 <sub>hex</sub>	(actually	expects
				256 <sub>dec</sub> )		

CLA	INS	P1	P2	Le		
00 <sub>hex</sub>	B0 <sub>hex</sub>	02 <sub>hex</sub>	00 <sub>hex</sub>	00 <sub>hex</sub>	(actually	expects
				256 <sub>dec</sub> )		

CLA	INS	P1	P2	Le		
00 <sub>hex</sub>	B0 <sub>hex</sub>	03 <sub>hex</sub>	00 <sub>hex</sub>	00 <sub>hex</sub>	(actually	expects
				256 <sub>dec</sub> )		

- c) For 5th, the last reading, there are A9<sub>hex</sub> bytes of certificate to read from the card from file offset 400<sub>hex</sub>:

CLA	INS	P1	P2	Le
00 <sub>hex</sub>	B0 <sub>hex</sub>	04 <sub>hex</sub>	00 <sub>hex</sub>	A9 <sub>hex</sub>

For other possible response messages see chapter 7.2.3 READ BINARY.

## 2.4. Cardholder secret keys

EstEID application has two PKI key pairs that enable cardholder authentication and digital signing operations. The length of the key is 2048<sub>dec</sub> bits. The public exponent for the keys is 40000081<sub>hex</sub> if supported by the platform or 00010001<sub>hex</sub> if the underlying platform is not capable of supporting arbitrary exponents. Initial remaining use counter for every key is FFFFFFFF<sub>hex</sub>. This counter will be decreased by one after every successful operation executed with the key. After the use counter has decreased to 0<sub>dec</sub>, the key can no longer be used.

Cardholder key pair cannot be read from the card. Key pairs are generated in the card on two occasions:

- Card application personalisation
- New key pair generation procedure

Information about the key can be read from EF with FID 0013<sub>hex</sub> in DF with FID EEEE<sub>hex</sub>. Given EF is variable-length formatted file and has 4 records containing information about key pairs as specified in following table:



**Table 2-4 EF FID 0013<sub>hex</sub> key records and references of secret keys**

Key description	Record no.	Reference no. KID    KV
Signature key (actively used)	01 <sub>hex</sub>	0100 <sub>hex</sub>
Signature key (can be absent)	02 <sub>hex</sub>	0200 <sub>hex</sub>
Authentication key (actively used)	03 <sub>hex</sub>	1100 <sub>hex</sub>
Authentication key (can be absent)	04 <sub>hex</sub>	1200 <sub>hex</sub>



The active keys could be changed during the life of the card and should be checked from EF FID 0033<sub>hex</sub> with procedure specified in chapter 2.4.2 Reading secret key information.

**Table 2-5 EF FID 0013<sub>hex</sub> key record description**

Bytes		Data	Description
Position	Count		
0 .. 1	2	8304 <sub>hex</sub>	Tag & Length for key reference
2 .. 3	2	XXXX <sub>hex</sub>	Key reference value
4 .. 5	2	0000 <sub>hex</sub>	Fixed value
6 .. 7	1	C002 <sub>hex</sub>	Tag & Length for public key data
8	1	81 <sub>hex</sub> initialised key	Public key indicator
		00 <sub>hex</sub> not initialised key	No key attached
9	1	FF <sub>hex</sub> initialised key	Public key size 2048 <sub>bits</sub>
		00 <sub>hex</sub> not initialised key	No key attached
10 .. 11	2	9103 <sub>hex</sub>	Tag & Length for key use counter
12 .. 14	3	XXXXXX <sub>hex</sub>	Key use counter value

The information about the currently active key pair can be read from the EF with FID 0033<sub>hex</sub> in DF with FID EEEE<sub>hex</sub>. Given EF is variable-length formatted file, but contains only 1 record. The structure of the only record of the file is specified in the following table:

**Table 2-6 EF FID 0033<sub>hex</sub> key record description**

Bytes		Data	Description
Position	Count		
0	1	00 <sub>hex</sub>	Security environment (SE) indicator. Applies to all SEs.
1 .. 2	2	A408 <sub>hex</sub>	Tag & Length of active authentication key info

Table 2-6 EF FID 0033 <sub>hex</sub> key record description			
Bytes		Data	Description
Position	Count		
3 .. 4	2	9501 <sub>hex</sub>	Tag & Length of authentication key usage qualifier
5	1	40 <sub>hex</sub>	Value for authentication key usage qualifier.
6 .. 7	2	8303 <sub>hex</sub>	Tag & Length of authentication key accessing info
8	1	80 <sub>hex</sub>	Value for authentication key search type (KST)
9 .. 10	2	XXXX <sub>hex</sub>	Value for active authentication key reference: Key ID (KID)
11 .. 12	2	B608 <sub>hex</sub>	Tag & Length of active signature key info
13 .. 14	2	9501 <sub>hex</sub>	Tag & Length of signature key usage qualifier
15	1	40 <sub>hex</sub>	Value for signature key usage qualifier
16 .. 17	2	8303 <sub>hex</sub>	Tag & Length of signature key accessing info
18	1	80 <sub>hex</sub>	Value for signature key search type (KST).
19 .. 20	2	XXXX <sub>hex</sub>	Value for active signature key reference: Key ID (KID)



Value 40<sub>hex</sub> of usage qualifier returned in response is specified in ISO 7816-9. It indicates the given key to be used for data authentication, data confidentiality, internal and mutual authentication.

Value 80<sub>hex</sub> of key search type indicates that the key is usable only in specific DF.

#### 2.4.1. Reading public key of cardholder secret key

There is only one way to obtain the public key from the card application, which is reading the certificate file and divide the public key from it.



This specification does not specify the procedure how to use or derive the public key from the certificate. This functionality must be searched and used by some cryptography API.

#### 2.4.2. Reading secret key information

The secret key information can be obtained by reading records from EF with FID 0013<sub>hex</sub>. The structure of given EF is specified in table EF FID 0013<sub>hex</sub> key records and references of secret keys.

EF FID 0013<sub>hex</sub> file is located in DF FID EEEE<sub>hex</sub>. First navigate to given directory by:

- a) selecting the application MF with command SELECT FILE:

CLA	INS	P1	P2	Le
00 <sub>hex</sub>	A4 <sub>hex</sub>	00 <sub>hex</sub>	0C <sub>hex</sub>	00 <sub>hex</sub>





b) and selecting directory file  $EEEE_{hex}$  by using command SELECT FILE again.

CLA	INS	P1	P2	Lc	Data (FID)
$00_{hex}$	$A4_{hex}$	$01_{hex}$	$0C_{hex}$	$02_{hex}$	$EEEE_{hex}$

Next step is to select EF FID  $0013_{hex}$  for reading by using command SELECT FILE once again:

CLA	INS	P1	P2	Lc	Data (FID)
$00_{hex}$	$A4_{hex}$	$02_{hex}$	$0C_{hex}$	$02_{hex}$	$0013_{hex}$

Now EF is ready for reading operations:

1) To read information for actively in use signature key from record no 1:

CLA	INS	P1 (record no)	P2	Le
$00_{hex}$	$B2_{hex}$	$01_{hex}$	$04_{hex}$	$00_{hex}$

2) To read information for secondary signature key from record no 2:

CLA	INS	P1 (record no)	P2	Le
$00_{hex}$	$B2_{hex}$	$02_{hex}$	$04_{hex}$	$00_{hex}$

3) To read information for actively in use authentication key from record no 3:

CLA	INS	P1 (record no)	P2	Le
$00_{hex}$	$B2_{hex}$	$03_{hex}$	$04_{hex}$	$00_{hex}$

4) To read information for secondary authentication key from record no 4:

CLA	INS	P1 (record no)	P2	Le
$00_{hex}$	$B2_{hex}$	$04_{hex}$	$04_{hex}$	$00_{hex}$

For successful operation, card application will respond to these commands in data field as described in table EF FID  $0013_{hex}$  key record description. For other possible R-APDUs see chapter 7.2.2 READ RECORD.

### 2.4.3. Reading key references for active keys

Key references for active keys can be read from EF FID  $0033_{hex}$  in record no 1.

Before given EF can be read it is needed to be selected. First navigate to given directory by:

1) selecting the application MF with command SELECT FILE:

CLA	INS	P1	P2	Le
$00_{hex}$	$A4_{hex}$	$00_{hex}$	$0C_{hex}$	$00_{hex}$

2) and selecting directory file  $EEEE_{hex}$  by using command SELECT FILE again.

CLA	INS	P1	P2	Lc	Data (FID)
$00_{hex}$	$A4_{hex}$	$01_{hex}$	$0C_{hex}$	$02_{hex}$	$EEEE_{hex}$

Next step is to select EF FID 0033<sub>hex</sub> for reading by using command SELECT FILE once again:

CLA	INS	P1	P2	Lc	Data (FID)
00 <sub>hex</sub>	A4 <sub>hex</sub>	02 <sub>hex</sub>	0C <sub>hex</sub>	02 <sub>hex</sub>	0033 <sub>hex</sub>

Now EF is ready for reading operations. Read record no 1 from selected file:

CLA	INS	P1 (record no)	P2	Le
00 <sub>hex</sub>	B2 <sub>hex</sub>	01 <sub>hex</sub>	04 <sub>hex</sub>	00 <sub>hex</sub>

For successful operation, card application will respond to these commands in data field as described in table EF FID 0033<sub>hex</sub> key record description. For other possible R-APDUs see chapter 7.2.2 READ RECORD.



The references for active signature and authentication keys EF FID 0033<sub>hex</sub> may be changed in the course of a new key pair generation operation (see chapter 4.4 New RSA key pair generation).

## 2.5. Card application management keys: CMK\_PIN, CMK\_CERT & CMK\_KEY

The security of EstEID card application management, for after personalisation process, is based on three secret 3DES keys:

- CMK\_PIN – Cardholder authentication objects replacement key. Used to get authorisation for executing command SECURE REPLACE PINS.
- CMK\_CERT – Cardholder certificate objects replacement key. Used to get authorisation for executing command SECURE REPLACE CERTIFICATE.
- CMK\_KEY – Cardholder new secret keys generation key. Used to get authorisation for executing command SECURE GENERATE KEY.

Each of these keys is derived from the corresponding master 3DES key by the card personalisation service provider in a secure environment. The given derived keys are loaded on the card application during card application personalisation phase and cannot be read from the card or modified afterwards.



The example keys and cardholder ID used in the present document are as follows:

- Master CMK\_PIN =  
A1A1A1A1A1A1A1A1A2A2A2A2A2A2A2A2<sub>hex</sub>
- Master CMK\_CERT =  
C1C1C1C1C1C1C1C1C2C2C2C2C2C2C2C2<sub>hex</sub>
- Master CMK\_KEY =  
B1B1B1B1B1B1B1B1B2B2B2B2B2B2B2B2<sub>hex</sub>
- Cardholder ID = “47101010033”

### 2.5.1. Deriving card application management keys

Each CMK key on card application is derived from corresponding master CMK key which are maintained by the card centre. Current chapter describes the procedure to derive CMK keys.

The CMK derivation procedure is as follows:

- 1) Calculate the SHA-1 hash of the cardholder's personal identification number.
- 2) Take 16 leftmost bytes of calculated SHA-1.
- 3) Encrypt these bytes with master CMK key, which correspond with desired one, by using 3DES algorithm, in CBC mode with IV value 0000000000000000<sub>hex</sub>.
- 4) Set the MSB of each byte off (set to zero)



For following example let the cardholder identification number be '01234567890'. For example calculation of the CMK there will be used master CMK given in chapter 2.5 Card application management keys: CMK\_PIN, CMK\_CERT & CMK\_

Example of calculating CMK\_PIN:

- 1) SHA-1 hash for given cardholder identification number is:  
7ED10E4A589C87F9E6A85C22E4B0C38ECF5F5059<sub>hex</sub>
- 2) 16 leftmost bytes of given hash:  
7ED10E4A589C87F9E6A85C22E4B0C38E<sub>hex</sub>
- 3) Encrypt bytes above with master CMK\_PIN in 3DEC CMC mode with IV value 0000000000000000<sub>hex</sub>:  
41F9AE3548536F19B93FED4EF890C93B<sub>hex</sub>
- 4) Set the LSB bit of each byte off:

Bytes as bit array:

```
01000001 11111001 10101110 00110101 01001000 01010011 01101111
00011001 10111001 00111111 11101101 01001110 11111000 10010000
11001001 00111011
```

Bytes as bit array with LSB set off:

```
01000000 11111000 10101110 00110100 01001000 01010010 01101110
00011000 10111000 00111110 11101100 01001110 11111000 10010000
11001000 00111010
```

Bytes with LSB set off:

CMK\_PIN = 40F8AE3448526E18B83EEC4EF890C83A<sub>hex</sub>

As well derivations for CMK\_CERT and CMK\_KEY:

CMK\_CERT = 3A8ABC9A981E28AAB20C961464284262<sub>hex</sub>

CMK\_KEY = 88FA5C9AB082D096AA125EBE70DEFC86<sub>hex</sub>

### 2.6. Miscellaneous information

From the application there can be read information that is not directly related to application data objects. There are 3 types of information available to read:

- EstEID card application version
- CPLC data
- Chip available memory

This information can be accessed with executing command GET DATA. Given information is not related to application file system and does not need navigation to data files.



For following commands Le field is optional if T1 protocol is used.

For protocol T0 Le field must be set. Otherwise the card will respond with 61XX<sub>hex</sub> or 6CXX<sub>hex</sub>. How to behave on given cases is specified in chapter [7.1 Card possible response in case of protocol T0](#).

### 2.6.1. Reading EstEID application version

To read EstEID card application version it is needed to executed the following command GET DATA:

CLA	INS	P1	P2	Le
00 <sub>hex</sub>	CA <sub>hex</sub>	01 <sub>hex</sub>	00 <sub>hex</sub>	03 <sub>hex</sub>



EstEID 3.5.4 card responds with only Major.Minor numbers: 3.5. EstEID 3.5.7 responds also patch number.



For given example let the EstEID application versin be 3.5.7

Card application will respond with following R-APDU containing data for EstEID application version:

Data	SW1	SW2
030507 <sub>hex</sub>	90 <sub>hex</sub>	00 <sub>hex</sub>

### 2.6.2. Reading CPLC data

To read CPLC data it is needed to execute the following command [GET DATA](#):

CLA	INS	P1	P2	Le
00 <sub>hex</sub>	CA <sub>hex</sub>	02 <sub>hex</sub>	00 <sub>hex</sub>	2A <sub>hex</sub>

Card application will respond with following R-APDU containing data for EstEID application version:

Data	SW1	SW2
42 <sub>dec</sub> bytes of CPLC data	90 <sub>hex</sub>	00 <sub>hex</sub>

The contents of the CPLC data sequence is specified in the following table:

Table 2-7 Card Production Life Cycle data	
CPLC field	Length
IC Fabricator	2 <sub>dec</sub>
IC Type	2 <sub>dec</sub>
Operating System Identifier	2 <sub>dec</sub>

**Table 2-7 Card Production Life Cycle data**

CPLC field	Length
Operating System release date	2 <sub>dec</sub>
Operating System release level	2 <sub>dec</sub>
IC Fabrication Date	2 <sub>dec</sub>
IC Serial Number	4 <sub>dec</sub>
IC Batch Identifier	2 <sub>dec</sub>
IC Module Fabricator	2 <sub>dec</sub>
IC Module Packaging Date	2 <sub>dec</sub>
ICC Manufacturer	2 <sub>dec</sub>
IC Embedding Date	2 <sub>dec</sub>
IC Pre-personaliser	2 <sub>dec</sub>
IC Pre-personalisation Date	2 <sub>dec</sub>
IC Pre-personalisation Equipment Identifier	4 <sub>dec</sub>
IC Personaliser	2 <sub>dec</sub>
IC Personalisation Date	2 <sub>dec</sub>
IC Personalisation Equipment Identifier	4 <sub>dec</sub>

### 2.6.3. Reading data for available memory on chip

To read free memory of chip it is needed to execute the following command GET DATA:

CLA	INS	P1	P2	Le
00 <sub>hex</sub>	CA <sub>hex</sub>	03 <sub>hex</sub>	00 <sub>hex</sub>	06 <sub>hex</sub>

Card application will respond with following R-APDU containing data for available memory on chip:

Data (Available memory...)			SW1	SW2
... that is freed on application deselecting or reset.	.. that is freed on application reset.	... that is for persistent use.		
XXXX <sub>hex</sub>	YYYY <sub>hex</sub>	ZZZZ <sub>hex</sub>	90 <sub>hex</sub>	00 <sub>hex</sub>



If there is more free memory available than FFFF<sub>hex</sub> bytes then given memory value will be returned as FFFF<sub>hex</sub>.



Command given in current chapter responds with sequence of results of JavaCard v2.2.2 API command JCSYSTEM.getAvailableMemory. The size of the free memory for three different types of memory in the card are returned. These

commands are executed with attributes in following order:

- MEMORY\_TYPE\_TRANSIENT\_DESELECT
- MEMORY\_TYPE\_TRANSIENT\_RESET
- MEMORY\_TYPE\_PERSISTENT

Look JavaCard v2.2.2 API for better overview.

### 3. Card application general operations

Card application enables PKI operations. Following sub-chapters describe how to perform operations regarding to cardholder authentication, digital signing and session key decryption.

#### 3.1. Calculating the response for TLS challenge

Calculating the response for TLS challenge is executing the RSA encryption operation with private key. Encrypted TLS challenge is formatted according to PKCS#1 version 1.5 block type 1. For authorising cardholder for given operation it is needed to authenticate the user with PIN1.



For given example let the PIN1 code have the same value as in chapter [2.2.1](#) Verify PIN1, PIN2 or PUK code.



This are just examples showing one possible way of doing it. Please refer to ISO7816-4 to exploit other methods and solutions.

- 1) In order to calculate the response by card application it is needed to execute following operations: Before currently selected key reference can be set it is needed to select DF FID  $EEEE_{hex}$ . First navigate to given directory by:

- a) selecting the application MF with command SELECT FILE:

CLA	INS	P1	P2	Le
$00_{hex}$	$A4_{hex}$	$00_{hex}$	$0C_{hex}$	$00_{hex}$

- b) and selecting directory file  $EEEE_{hex}$  by using command SELECT FILE again.

CLA	INS	P1	P2	Lc	Data (FID)
$00_{hex}$	$A4_{hex}$	$01_{hex}$	$0C_{hex}$	$02_{hex}$	$EEEE_{hex}$

- 2) Select security environment for TLS operations by executing following command MANAGE SECURITY ENVIRONMENT:

CLA	INS	P1	P2	Le
$00_{hex}$	$22_{hex}$	$F3_{hex}$	$01_{hex}$	$00_{hex}$



If the card has already been set to use decryption security environment

after last card reset, then it is not required to execute given command.

- 3) Set the active authentication key reference for INTERNAL AUTHENTICATE command execution by executing command MANAGE SECURITY ENVIRONMENT:

CLA	INS	P1	P2	Lc	Data (TL of TLV    KST    Key reference)
00 <sub>hex</sub>	22 <sub>hex</sub>	41 <sub>hex</sub>	B8 <sub>hex</sub>	05 <sub>hex</sub>	8303 <sub>hex</sub>    80 <sub>hex</sub>    1100 <sub>hex</sub>



If the card hasn't been set to use secondary authentication key pair after last card reset, then it is not required to execute given command.



Key reference values as specified in Table 3-1 EF FID 0013hex key records and references of secret keys active keys should be used. The proper way to receive currently active authentication keys reference is described in chapter 2.4.3 Reading key references for active keys.



By default it is not necessary to execute the security environment and key reference commands in case of calculating response to TLS challenge.

- 4) Authenticate cardholder with PIN1 with executing command VERIFY to authorise executing command INTERNAL AUTHENTICATE:

CLA	INS	P1	P2	Lc	Data (PIN1 as ASCII)
00 <sub>hex</sub>	20 <sub>hex</sub>	00 <sub>hex</sub>	01 <sub>hex</sub>	04 <sub>hex</sub>	31323334 <sub>hex</sub>

- 5) Calculate the response for TLS challenge by executing command INTERNAL AUTHENTICATE:

CLA	INS	P1	P2	Lc	Data	Le
00 <sub>hex</sub>	88 <sub>hex</sub>	00 <sub>hex</sub>	00 <sub>hex</sub>	TLS challenge length	TLS challenge	00 <sub>hex</sub>

Card application responds with bytes which are the cryptogram of PKCS#1 ver. 1.5 block type 1 enveloped TLS challenge. The length of the response varies and is related to the length of provided TLS challenge. The encryption is done with RSA authentication private key.

According to TLS standard, the length of the challenge is 24<sub>hex</sub> bytes. However, card application also enables the calculation of responses to challenges of other length. For the maximum length of TLS challenge it must be taken into account that it is being formatted according to PKCS#1 ver. 1.5 before cryptographic operation which requires 11<sub>dec</sub> bytes from the length of input data. Therefore the maximum length of TLS challenge, which can be used, is F5<sub>hex</sub> bytes.

## 3.2. Calculating the electronic signature

The calculation of electronic signature is the encryption of a hash object which will be formatted according to PKCS#1 ver. 1.5 block type 1.

Card application enables the calculation of the electronic signature using the RSA key in two ways:

- Providing card application with already calculated hash for signing procedure.
- Providing card application with data to be hashed before the signature procedure.

The result of signing procedure is cryptogram of PKCS#1 ver. 1.5 block type 1 enveloped hash. The encryption is done with RSA signature private key.

### 3.2.1. Calculating the electronic signature with providing pre-calculated hash

Current chapter describes the signing method where hash is calculated by the host application and provided to card application prior to signing operation. For authorising cardholder for given operation it is needed to authenticate the user with PIN2.



For given example let the PIN2 code have the same value as in chapter 2.2.1 Verify PIN1, PIN2 or PUK code.

Let the active signature private key reference be 0100<sub>hex</sub> for given example.

To calculate electronic signature for given method, following procedures should be executed:

- 1) Before currently selected key reference can be set it is needed to select DF FID EEEE<sub>hex</sub>. First navigate to given directory by:

- a) selecting the application MF with command SELECT FILE:

CLA	INS	P1	P2	Le
00 <sub>hex</sub>	A4 <sub>hex</sub>	00 <sub>hex</sub>	0C <sub>hex</sub>	00 <sub>hex</sub>

- b) and selecting directory file EEEE<sub>hex</sub> by using command SELECT FILE again.

CLA	INS	P1	P2	Lc	Data (FID)
00 <sub>hex</sub>	A4 <sub>hex</sub>	01 <sub>hex</sub>	0C <sub>hex</sub>	02 <sub>hex</sub>	EEEE <sub>hex</sub>

- 2) Select security environment for signature calculation operations by executing following command MANAGE SECURITY ENVIRONMENT:

CLA	INS	P1	P2	Le
00 <sub>hex</sub>	22 <sub>hex</sub>	F3 <sub>hex</sub>	01 <sub>hex</sub>	00 <sub>hex</sub>

- 3) Set the key reference to active signature key for COMPUTE DIGITAL SIGNATURE command execution by executing command MANAGE SECURITY ENVIRONMENT:

CLA	INS	P1	P2	Lc	Data (TL of TLV    KST    Key reference)
00 <sub>hex</sub>	22 <sub>hex</sub>	41 <sub>hex</sub>	B8 <sub>hex</sub>	05 <sub>hex</sub>	8303 <sub>hex</sub>    80 <sub>hex</sub>    0100 <sub>hex</sub>



If the card hasn't been set to use secondary signature key pair after last card reset, then it is not required to execute given command.





Key reference values as specified in Table 3-2 EF FID 0013hex key records and references of secret keys active keys should be used. The proper way to receive currently active signature key reference is described in chapter 2.4.3 Reading key references for active keys.



By default it is not necessary to execute the security environment and key reference commands in case of calculating the digital signature with pre-calculated hash.

- 4) Authorise cardholder for executing command COMPUTE DIGITAL SIGNATURE by authenticating user with PIN2 by executing command VERIFY:

CLA	INS	P1	P2	Lc	Data (PIN2 as ASCII)
00 <sub>hex</sub>	20 <sub>hex</sub>	00 <sub>hex</sub>	02 <sub>hex</sub>	05 <sub>hex</sub>	3132333435 <sub>hex</sub>

- 5) Calculate the electronic signature by executing command COMPUTE DIGITAL SIGNATURE:

CLA	INS	P1	P2	Lc	Data
00 <sub>hex</sub>	2A <sub>hex</sub>	9E <sub>hex</sub>	9A <sub>hex</sub>	Data length	Hash algorithm identifier    Data of hash

The following list specifies supported hash algorithms and their identifiers:

- SHA-1: 3021300906052B0E03021A05000414<sub>hex</sub>
- SHA-224: 302D300D06096086480165030402040500041C<sub>hex</sub>
- SHA-256: 3031300D060960864801650304020105000420<sub>hex</sub>
- SHA-384: 3041300D060960864801650304020205000430<sub>hex</sub>
- SHA-512: 3051300D060960864801650304020305000440<sub>hex</sub>



The support of SHA-384 and SHA-512 hash function are optional and depend on the support of the underlying chip, the integrated circuit and the java operating system.



Data transmitted to card is not validated by the card application. In case of invalid data format the card application can respond with response codes described in chapter 7.3 Error response APDU messages.

For successful operation the card application responds with signature RSA private key encrypted PKCS#1 ver. 1.5 block type 1 wrapped data. For card application possible responses see chapter 7.2.13.3 COMPUTE DIGITAL SIGNATURE.

### 3.2.2. Calculating the electronic signature with internal hash calculating



This feature is deprecated for EstEID v3.5.7 and higher versions.

Current chapter describes the signing method where SHA-1 hash will be calculated by the card application prior to signing operation. This kind of procedure may be needed for

POS devices which do not have SHA-1 hashing functionality. For authorising cardholder for given operation it is needed to authenticate the user with PIN2.



For given example let the PIN2 code have the same value as in chapter 2.2.1 Verify PIN1, PIN2 or PUK code.

Let the active signature private key reference be 0100<sub>hex</sub> for given example.

To calculate electronic signature for given method, following procedures should be executed:

- 1) Before currently selected key reference can be set it is needed to select DF FID EEEE<sub>hex</sub>. First navigate to given directory by:

- a) selecting the application MF with command SELECT FILE:

CLA	INS	P1	P2	Le
00 <sub>hex</sub>	A4 <sub>hex</sub>	00 <sub>hex</sub>	0C <sub>hex</sub>	00 <sub>hex</sub>

- b) and selecting directory file EEEE<sub>hex</sub> by using command SELECT FILE again.

CLA	INS	P1	P2	Lc	Data (FID)
00 <sub>hex</sub>	A4 <sub>hex</sub>	01 <sub>hex</sub>	0C <sub>hex</sub>	02 <sub>hex</sub>	EEEE <sub>hex</sub>

- 2) Select security environment for signature calculation operations by executing following command MANAGE SECURITY ENVIRONMENT:

CLA	INS	P1	P2	Le
00 <sub>hex</sub>	22 <sub>hex</sub>	F3 <sub>hex</sub>	01 <sub>hex</sub>	00 <sub>hex</sub>

- 3) Set the key reference for COMPUTE DIGITAL SIGNATURE command execution by executing command MANAGE SECURITY ENVIRONMENT:

CLA	INS	P1	P2	Lc	Data (TL of TLV    KST    Key reference)
00 <sub>hex</sub>	22 <sub>hex</sub>	41 <sub>hex</sub>	B8 <sub>hex</sub>	05 <sub>hex</sub>	8303 <sub>hex</sub>    80 <sub>hex</sub>    0100 <sub>hex</sub>



If the card hasn't been set to use secondary signature key pair after last card reset, then it is not required to execute given command.



Key reference values as specified in Table 3-3 EF FID 0013hex key records and references of secret keys active keys should be used. The proper way to receive currently active signature key reference is described in chapter 2.4.3 Reading key references for active keys.

- 4) Authorise cardholder for executing command COMPUTE DIGITAL SIGNATURE by authenticating user with PIN2 by executing command VERIFY:

CLA	INS	P1	P2	Lc	Data (PIN2 as ASCII)
00 <sub>hex</sub>	20 <sub>hex</sub>	00 <sub>hex</sub>	02 <sub>hex</sub>	05 <sub>hex</sub>	3132333435 <sub>hex</sub>

- 5) Transmit data to card application for SHA-1 hashing by executing command HASH:

CLA	INS	P1	P2	Lc	Data
00 <sub>hex</sub>	2A <sub>hex</sub>	90 <sub>hex</sub>	A0 <sub>hex</sub>	Data length	SHA-1 identifier    Data for hashing

For successful operation, of current command, SHA-1 hash will be returned in R-APDU and as well the same hash is held inside the card application for using by the following command COMPUTE DIGITAL SIGNATURE.



If the field data length exceeds the length of normal APDU, command HASH must be transmitted as chained or extended. APDU chaining is described in chapter 7.4 Message chaining and extended APDU usage is described in chapter 7.5 Extended APDU.

- 6) Calculate the electronic signature by executing command COMPUTE DIGITAL SIGNATURE:

CLA	INS	P1	P2	Le
00 <sub>hex</sub>	2A <sub>hex</sub>	9E <sub>hex</sub>	9A <sub>hex</sub>	00 <sub>hex</sub>

For successful operation the card application responds with signature RSA private key encrypted PKCS#1 ver. 1.5 block type 1 wrapped data. For card application possible responses see chapter 7.2.13.3 COMPUTE DIGITAL SIGNATURE.

### 3.3. Decrypting public key encrypted data

Current chapter describes the decrypting method which must be used to execute RSA decryption operation with private key on data which is encrypted with the corresponding authentication public key. Encrypted data must be formatted according to PKCS#1 version 1.5 block type 2. For authorising cardholder for given operation it is needed to authenticate the user with PIN1.



For given example let the PIN1 code have the same value as in chapter 2.2.1 Verify PIN1, PIN2 or PUK code.

Let the active and secondary authentication private key references be respectively 1100<sub>hex</sub> and 1200<sub>hex</sub> for given example.

To decrypt cryptogram of PKCS#1 ver. 1.5 block type 2 enveloped data, following procedures should be executed:

- 1) Before currently selected key reference can be set it is needed to select DF FID EEEE<sub>hex</sub>. First navigate to given directory by:
  - a) selecting the application MF with command SELECT FILE:

CLA	INS	P1	P2	Le
00 <sub>hex</sub>	A4 <sub>hex</sub>	00 <sub>hex</sub>	0C <sub>hex</sub>	00 <sub>hex</sub>

- b) and selecting directory file  $EEEE_{hex}$  by using command SELECT FILE again.

CLA	INS	P1	P2	Lc	Data (FID)
$00_{hex}$	$A4_{hex}$	$01_{hex}$	$0C_{hex}$	$02_{hex}$	$EEEE_{hex}$

- 2) Select security environment for decryption operations by executing following command MANAGE SECURITY ENVIRONMENT:

CLA	INS	P1	P2	Le
$00_{hex}$	$22_{hex}$	$F3_{hex}$	$06_{hex}$	$00_{hex}$



If the card has already been set to use decryption security environment after last card reset, then it is not required to execute given command.

- 3) Set the key reference for DECIPHER command execution by executing command MANAGE SECURITY ENVIRONMENT:

or

- a) For decrypting with private authentication key that has reference value  $1100_{hex}$ , use command:

CLA	INS	P1	P2	Lc	Data (TL of TLV    KST    Key reference)
$00_{hex}$	$22_{hex}$	$41_{hex}$	$A4_{hex}$	$05_{hex}$	$8303_{hex}    80_{hex}    1100_{hex}$



If the card hasn't been set to use secondary authentication key pair after last card reset, then it is not required to execute given command.

- b) For decrypting with private authentication key that has reference value  $1200_{hex}$ , use command:

CLA	INS	P1	P2	Lc	Data (TL of TLV    KST    Key reference)
$00_{hex}$	$22_{hex}$	$41_{hex}$	$A4_{hex}$	$05_{hex}$	$8303_{hex}    80_{hex}    1200_{hex}$



Key reference values as specified in Table 3-4 EF FID 0013hex key records and references of secret keys active keys should be used. The proper way to receive currently active authentication keys reference is described in chapter 2.4.3 Reading key references for active keys.



It is not possible to use Signature key for deciphering operations. Only authentication keys can be used for this procedure.

- 4) Authorise cardholder for executing command DECIPHER by authenticating user with executing command VERIFY for verifying PIN1:

CLA	INS	P1	P2	Lc	Data (PIN1 as ASCII)
00 <sub>hex</sub>	20 <sub>hex</sub>	00 <sub>hex</sub>	01 <sub>hex</sub>	04 <sub>hex</sub>	31323334 <sub>hex</sub>



For the following operation it must be sure that decrypted data is formatted according to PKCS#1 ver. 1.5 block type 2 and the cryptogram is the result of encryption with RSA authentication public key. Otherwise the result will be an error situation.



The encrypted data transmitted to the card application must be pre-padded with 00<sub>hex</sub> byte which indicates that it is formatted according to PKCS#1 ver. 1.5 block type 2.

- 5) Decrypt and obtain session key by executing command DECIPHER:

CLA	INS	P1	P2	Lc	Data
00 <sub>hex</sub>	2A <sub>hex</sub>	80 <sub>hex</sub>	86 <sub>hex</sub>	Data length	Data for RSA public key encrypted session key



If the length of the cryptogram for command DECIPHER exceeds the data length of normal APDU it must be transmitted as chained or extended. APDU chaining is described in chapter 7.4 Message chaining and extended APDU usage is described in chapter 7.5 Extended APDU.

For successful operation the card application responds with plain data unwrapped from the PKCS#1 ver. 1.5 block type 2 envelope. For card application possible responses see chapter 7.2.13.2 DECIPHER.

## 4. Card application managing operations

Current chapter describes procedures for the case if there comes up a need to replace PKI objects on the card. These procedures are not meant to be performed by any other institution than card authority.

### 4.1. Secure channel communication

In terms of card application managing operations it is required to hold transmission channel secured. All messages are secured with 3DES session keys by encrypting data and calculating signature for command.

C-APDUs which support secure channel are described in following table.

Table 4-1 C-APDUs supporting secure channel	
INS	Command
05 <sub>hex</sub>	Replace PINs/PUK



Table 4-1 C-APDUs supporting secure channel	
INS	Command
06 <sub>hex</sub>	Generate new key pair
07 <sub>hex</sub>	Replace certificate
B0 <sub>hex</sub>	Read Binary
B2 <sub>hex</sub>	Read Record
CD <sub>hex</sub>	Get Data

#### 4.1.1. Mutual Authentication

Mutual Authentication is operation where host application gets authorisation to access card application management operations. To get authorisation host application must authenticate itself to card application.

To authenticate the host application, following operations should be executed:

- 1) Get challenge (random number RND.ICC) from the card by executing command GET CHALLENGE:

CLA	INS	P1	P2	Le
00 <sub>hex</sub>	84 <sub>hex</sub>	00 <sub>hex</sub>	00 <sub>hex</sub>	08 <sub>hex</sub>

Card responds with 8<sub>dec</sub> bytes challenge:

Data	SW1	SW2
RND.ICC (8 <sub>dec</sub> bytes)	90 <sub>hex</sub>	00 <sub>hex</sub>

- 2) Authenticate host application by executing command MUTUAL AUTHENTICATE.

Before given command can be executed following operations must be performed:

- a) Generate host application 8 bytes of random data called RND.IFD.
- b) Generate 2\*16<sub>dec</sub> random data, called K.IFD, which is host application side data for session keys calculation.
- c) Concatenate together RND.IFD, RND.ICC and K.IFD in given order. You get 30<sub>hex</sub> of data for encryption.
- d) Derive cardholder CMK key which is related to procedure taking place after Mutual Authentication.
- e) Encrypt 30<sub>hex</sub> bytes data with derived cardholder CMK 3DES key using CBC mode.
- f) Use calculated cryptogram in command MUTUAL AUTHENTICATE data field and transmit to card application:

CLA	INS	P1	P2	Lc	Data	Le
00 <sub>hex</sub>	82 <sub>hex</sub>	00 <sub>hex</sub>	0X <sub>hex</sub>	30 <sub>hex</sub>	Cryptogram 30 <sub>hex</sub> bytes	30 <sub>hex</sub>

Successful operation response:

Data	SW1	SW2
Cryptogram 30 <sub>hex</sub> bytes	90 <sub>hex</sub>	00 <sub>hex</sub>

- g) Decrypt 30<sub>hex</sub> bytes received data with derived cardholder CMK 3DES key using CBC mode.

- h) Decrypted data contains components in following order: RND.ICC || RND.IFD || K.ICC.
- i) Verify received RND.IFD by comparing it to generated RND.IFD. They must match!
- j) Calculate session keys (SK) by applying XOR operation between K.ICC and K.IFD.
- k) Encryption session key (SK1) is the 16<sub>dec</sub> leftmost bytes of SK. Signature (MAC) session key (SK2) is the 16<sub>dec</sub> rightmost bytes of SK.
- l) Calculate IV called Send Sequence Counter (SSC) for following secured command executions. SCC is put together by concatenating 4 leftmost bytes of RND.IFD and 4 leftmost bytes of RND.ICC.

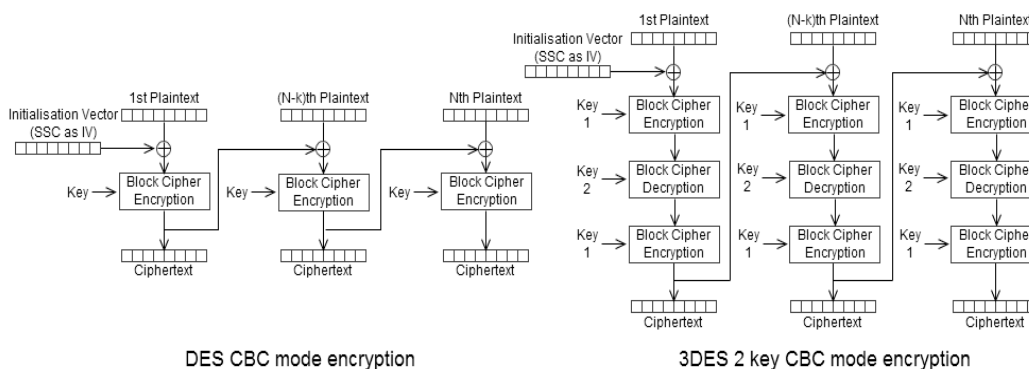
#### 4.1.2. Channel securing

Commands which are sent to card application as secured must have command data encrypted and command signature (MAC) appended to the command. These operations can be performed after successful authentication procedure described in chapter [4.1.1 Mutual Authentication](#).

Data encryption and command MAC calculation must be performed with 3DES key using CBC mode. Still there is a small difference between these operations. For MAC calculating operation it is needed to use only 8 leftmost bytes of the DES key using CBC mode for N-1 blocks. Only for the last Nth block encrypting with the full 3DES key using CBC mode is performed. For encryption operation normal 3DES key is used. See figure below for encryption with DES CBC and 3DES CBC.

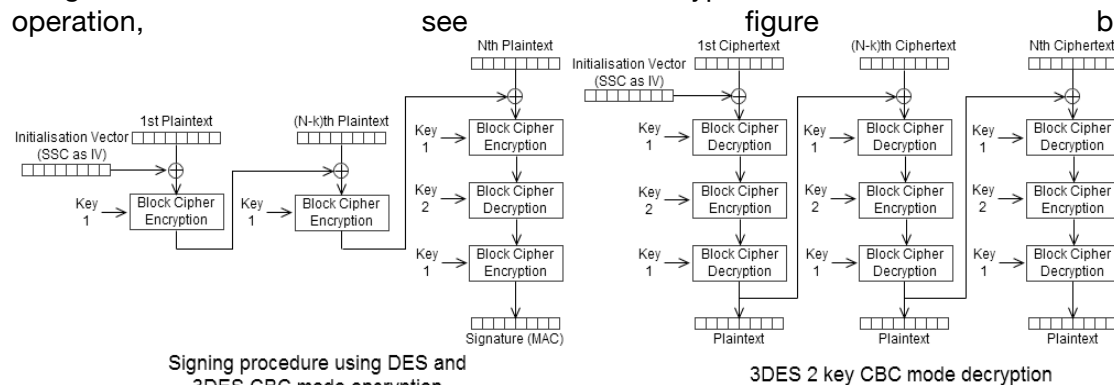


Data for MAC calculating and encryption operations must be padded according to ISO 9797-1 padding method 2. Given method tells to append 80<sub>hex</sub> byte and as many 00<sub>hex</sub> bytes to data until its length is modulus of DES algorithm block length, which is 8.



**Figure 4-1** DES and 3DES CBC mode encryption

To get the overview of 3DES CBC mode decryption and actual MAC calculating operation, see figure below.



**Figure 4-2** MAC signature calculation and 3DES CBC mode decryption



Specification for DES and TDES algorithms can be found in ISO 18033-3:2010.

To make secured C-APDU, following procedures must be performed:

- 1) Increase the value of SSC with one. If the value of SSC is  $\text{FFFFFFFFFFFFFFFF}_{\text{hex}}$  then after increasing operation it gets value  $\text{0000000000000000}_{\text{hex}}$ . See following examples:  
 $\text{FF73F9D201044A59}_{\text{hex}} + 1 = \text{FF73F9D201044A59}_{\text{hex}}$   
 $\text{FFFFFFFFFFFFFFFF}_{\text{hex}} + 1 = \text{0000000000000000}_{\text{hex}}$
- 2) Change C-APDU CLA value to  $\text{0C}_{\text{hex}}$ , which indicates, that the command is secured.
- 3) If there is data present in C-APDU:
  - a) Append padding to data according to ISO 9797-1 method 2.
  - b) Encrypt the data of C-APDU with SK1 in 3DES CBC mode using SSC value from previous step as IV.
  - c) Wrap cryptogram from previous step to TLV having tag  $\text{87}_{\text{hex}}$ , which identifies that the value is encrypted.
  - d) C-APDU data field must be switched with TLV from previous step.
- 4) Prepare data for MAC calculating. Get 4 header bytes of C-APDU which are CLA, INS, P1 and P2. Append  $\text{80000000}_{\text{hex}}$  bytes to it, so the result is 8 bytes in length.  
 $\text{CLA} \parallel \text{INS} \parallel \text{P1} \parallel \text{P2} \parallel \text{80000000}_{\text{hex}}$
- 5) If there is TLV wrapped cryptogram present in C-APDU, append it to data for MAC calculation.
- 6) Append padding to MAC calculating data according to ISO 9797-1 method 2.
- 7) Sign the data with encryption method described in figure MAC signature calculation and 3DES CBC mode decryption. In the figure Key 1 is the 8 leftmost and Key 2 is the 8 rightmost bytes of SK2. As IV the value of SSC must be used. Result of given encryption operation is 8 bytes of MAC signature.
- 8) Wrap MAC signature from previous step to TLV having tag  $\text{8E}_{\text{hex}}$ , which indicates that the value is the MAC signature.
- 9) Append TLV from previous step to C-APDU data field.



Now C-APDU is secured and can be transmitted to card application. The response from card application as well is secured and must be verified and data decrypted.

To verify and decrypt data of secured R-APDU, following procedures must be performed:

- 1) Increase the value of SSC with one.
- 2) Find MAC TLV with tag  $8E_{hex}$  from the  $10_{dec}$  leftmost bytes of R-APDU data field. Unwrap the value from this TLV to get MAC signature.
- 3) Prepare data for MAC calculation. Take R-APDU data field without MAC signature TLV.
- 4) Append padding to MAC calculating data according to ISO 9797-1 method 2.
- 5) Sign the data with encryption method described in figure MAC signature calculation and 3DES CBC mode decryption. In the figure Key 1 is the 8 leftmost and Key 2 is the 8 rightmost bytes of SK2. As ICV the value of SSC must be used. Result of given encryption operation is 8 bytes of MAC signature.
- 6) Verify R-APDU MAC by comparing it to calculated MAC. They must match!
- 7) If the R-APDU data without MAC signature TLV is TLV with tag...
  - a)  $\dots 99_{hex}$ , then the TLV value marks the 2 byte status word. It is MAC signed and therefore it can be sure about its integrity.
  - b)  $\dots 87_{hex}$ , then the TLV value is the cryptogram of actual R-APDU data and it needs to be decrypted.  
Decrypt the data of R-APDU with SK1 in 3DES CBC mode using SSC value from the first step as IV. Result of given decryption operation is the plaintext data of R-APDU.

## 4.2. PIN1, PIN2 and PUK replacement

In case the cardholder has forgotten or lost PIN/PUK codes, they can be replaced by the EstEID card authority.

To replace PIN/PUK codes, the following procedures must be performed:

- 1) Perform Mutual Authentication with cardholder CMK derived from CMK\_PIN.
  - 2) Replace cardholder PIN1, PIN2 and PUK codes by executing command REPLACE PINS (SECURE) as encrypted and MAC signed as described in chapter 4.1.2
- Channel securing:

CLA	INS	P1	P2	Lc
$0C_{hex}$	$05_{hex}$	$00_{hex}$	$00_{hex}$	$11_{hex}$

For successful operation card application responds with following encrypted and MAC signed R-APDU:

Data	SW1	SW2
empty	$90_{hex}$	$00_{hex}$



See APPENDIX chapter Replace cardholder PINs/PUK codes for example with secured messages and mutual authentication.

### 4.3. Certificate replacement

The validity period of the card may exceed the validity period of the certificates. In that case new certificates based on existing key pairs but with an extended validity period can be requested and loaded.

Current chapter describes the procedure to perform the replacement of cardholder certificate file.



For given example let the PIN1 code have the same values as in chapter 2.2.1 Verify PIN1, PIN2 or PUK code.

To replace certificates, the following procedures must be performed:

- 1) Perform Mutual Authentication with cardholder CMK derived from CMK\_CERT.
- 2) Verify cardholder with PIN1 by executing command VERIFY:

CLA	INS	P1	P2	Lc	Data (PIN1 as ASCII)
00 <sub>hex</sub>	20 <sub>hex</sub>	00 <sub>hex</sub>	01 <sub>hex</sub>	04 <sub>hex</sub>	31323334 <sub>hex</sub>

- 3) Replace current certificate with new one in the card application by executing command REPLACE CERTIFICATE (SECURE) as encrypted and MAC signed as described in chapter 4.1.2 Channel securing:

- a) To replace authentication certificate, execute:

CLA	INS	P1	P2	Lc	Data
0C <sub>hex</sub>	07 <sub>hex</sub>	01 <sub>hex</sub>	00 <sub>hex</sub>	0600 <sub>hex</sub>	

- b) To replace signature certificate, execute:

CLA	INS	P1	P2	Lc	Data
0C <sub>hex</sub>	07 <sub>hex</sub>	02 <sub>hex</sub>	00 <sub>hex</sub>	0600 <sub>hex</sub>	

For successful operation card application responds with following encrypted and MAC signed R-APDU:

Data (TLV formatted 271 <sub>dec</sub> bytes)	SW1	SW2
7F49 <sub>hex</sub>    82010A <sub>hex</sub>    81 <sub>hex</sub>    820100 <sub>hex</sub>    public key    82 <sub>hex</sub>    04 <sub>hex</sub>    exponent	90 <sub>hex</sub>	00 <sub>hex</sub>



See APPENDIX chapter Replace Certificates for example with secured messages and mutual authentication.

### 4.4. New RSA key pair generation

The request and loading of new certificates is not limited to the use of the active key pairs. New key pairs can be generated prior to the request and loading of new certificates.



For given example let the PIN1 code have the same values as in chapter 2.2.1 Verify PIN1, PIN2 or PUK code.

To generate new key pair, the following procedures must be performed:

- 1) Perform Mutual Authentication with cardholder CMK derived from CMK\_KEY.
- 2) Verify cardholder with PIN1 by executing command VERIFY:

CLA	INS	P1	P2	Lc	Data (PIN1 as ASCII)
00 <sub>hex</sub>	20 <sub>hex</sub>	00 <sub>hex</sub>	01 <sub>hex</sub>	04 <sub>hex</sub>	31323334 <sub>hex</sub>

- 3) Generate new key pair by executing command GENERATE KEY (SECURE) as encrypted and MAC signed as described in chapter 4.1.2 Channel securing:

- a) To generate new actively used authentication RSA key pair, execute:

CLA	INS	P1	P2	Le
0C <sub>hex</sub>	06 <sub>hex</sub>	01 <sub>hex</sub>	01 <sub>hex</sub>	00 <sub>hex</sub>

- b) To generate new actively used signature RSA key pair, execute:

CLA	INS	P1	P2	Le
0C <sub>hex</sub>	06 <sub>hex</sub>	01 <sub>hex</sub>	02 <sub>hex</sub>	00 <sub>hex</sub>

- c) To generate new secondary authentication RSA key pair, execute:

CLA	INS	P1	P2	Le
0C <sub>hex</sub>	06 <sub>hex</sub>	02 <sub>hex</sub>	01 <sub>hex</sub>	00 <sub>hex</sub>

- d) To generate new secondary signature RSA key pair, execute:

CLA	INS	P1	P2	Le
0C <sub>hex</sub>	06 <sub>hex</sub>	02 <sub>hex</sub>	02 <sub>hex</sub>	00 <sub>hex</sub>

After previous command active key references for corresponding authentication or signature key gets changed to one which was generated. The new reference value for currently active keys should be read from file with FID 0033<sub>hex</sub> as described in chapter 2.4.3 Reading key references for active keys.

For successful operation card application responds with following encrypted and MAC signed R-APDU:

Data (TLV formatted 271 <sub>dec</sub> bytes)	SW1	SW2
7F49 <sub>hex</sub>    82010A <sub>hex</sub>    81 <sub>hex</sub>    820100 <sub>hex</sub>    public key    82 <sub>hex</sub>    04 <sub>hex</sub>    exponent	90 <sub>hex</sub>	00 <sub>hex</sub>



See APPENDIX chapter Generate new key pair for example with secured messages and mutual authentication.

## 5. Card application security structure

Card application has three environments:

- Public environment – Reading data objects on the card.
- PKI environment – Needs PIN verification for operating.
- Card application authority environment – Using CMK secure messaging for operating.

All card operations and their access rights are showed in following table:

	Reading personal data file	Reading user certificates	Using Authentication key	Using signature key	PIN1 unblocking (also with PIN1 replacement)	PIN2 unblocking (also with PIN2 replacement)	Changing PIN1, PIN2 and PUK	Deciphering and authenticating operations	Digital signing operation	Unblocking and replacing PIN1, PIN2 and PUK	Replacing certificates	Generating new key pair
Public environment	ALW	ALW	NEV	NEV	PUK	PUK	PIN/PUK	NEV	NEV	NEV	NEV	NEV
PKI environment	NEV	NEV	PIN1	PIN2	NEV	NEV	NEV	PIN1	PIN2	NEV	NEV	NEV
Card application authority environment	ALW	ALW	NEV	NEV	NEV	NEV	NEV	NEV	NEV	ALW	PIN1	PIN1

Acronym	Description
ALW	Always allowed
NEV	Not allowed
PIN1	Operation can be used after PIN1 is verified
PIN2	Operation can be used after PIN2 is verified
PIN/PUK	Each PIN or PUK can be changed with verifying corresponding current PIN or PUK

## 6. Card application constants

Some of the objects that are set on the card in personalisation phase cannot be manipulated afterwards. These constant values concern the maximum and minimum values of object and fixed object values.

Table 6-1 Card application constant values					
Length	Length in bytes			Initial value	Fixed value
	Min	Max	Initial value		
PIN1	4	12 <sub>dec</sub>	4	From personalisation	
PIN2	5	12 <sub>dec</sub>	5		
PUK	8	12 <sub>dec</sub>	8		
RSA	256 <sub>dec</sub>	256 <sub>dec</sub>	256 <sub>dec</sub>	Generated in personalisation	
RSA public exponent	4	4	4	40000081 <sub>hex</sub>	40000081 <sub>hex</sub>
Certificate file	600 <sub>hex</sub>	600 <sub>hex</sub>	600 <sub>hex</sub>	From personalisation	
PIN/PUK retry counter	1	1	1	03 <sub>hex</sub>	
RSA usage countdown	3	3	3	FFFFFF <sub>hex</sub>	
CMK_PIN	16 <sub>dec</sub>	16 <sub>dec</sub>	16 <sub>dec</sub>	From personalisation	From personalisation
CMK_CERT	16 <sub>dec</sub>	16 <sub>dec</sub>	16 <sub>dec</sub>		
CMK_KEY	16 <sub>dec</sub>	16 <sub>dec</sub>	16 <sub>dec</sub>		



Lengths specified in chapter 2.1 Personal data file in table Personal Data file contents, should be taken as a part of current chapter.



The exponent of RSA keys is either 40000081<sub>hex</sub>, if supported by the platform, or 00010001<sub>hex</sub>, if the underlying platform is not capable of supporting arbitrary exponents.

## 7. APDU protocol

Communication between a chip card and a host application is performed over application-level APDU protocol. Current chapter and subchapters gives the basics of APDU protocol usage. APDU protocol itself is specified in ISO 7816-4 standard.

APDU messages compromise two structures: one used by

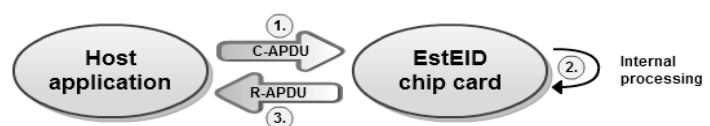


Figure 7-1 APDU master-slave communication

the host application to send commands to the card whereas one is used by the chip to send command response back to the host application. Data transmission between two ends is performed as master-slave communication where a host application is the master and a chip is the slave.

Command sent by a host application is called Command APDU (C-APDU) or simply APDU. Command sent by the chip as a response to C-APDU is called Response APDU (R-APDU).

APDU messages can be transmitted with two different transmission-level Transmission Protocol Data Units (TPDU) which are T0 and T1. The T0 and T1 protocols are used to support APDU protocols transmission between chip reader and chip itself. APDU protocol is used between the chip application and the chip reader.

T1 is block oriented protocol which enables blocks or grouped collections of data to be transferred. These data groups are transferred as a whole between chip and reader. The theoretical maximum length of T1 grouped collections for C-APDU is 65535<sub>dec</sub> and for R-APDU is 65536<sub>dec</sub> bytes. The practical maximum length depends on the chip platform that is used for EstEID application.

T0 is byte oriented protocol which means that the minimum data that can be transferred has a length of one byte. The maximum length of data structure that can be transferred with this protocol for C-APDU is 255<sub>dec</sub> and for R-APDU is 256<sub>dec</sub> bytes.



APDU structure defined in ISO 7816-4 standard is very similar to TDPU structure used in T0. When APDU is transmitted with T0, the elements of APDU exactly overlay the elements of TPDU.

Table 7-1 C-APDU structure						
Header				Body		
CLA	INS	P1	P2	Lc	Data	Le
				Optional for T1		
				Optional for T0		

Table 7-2 C-APDU contents			
Code	Name	Length	Description
CLA	Class	1	Class of instruction
INS	Instruction	1	Instruction code. Basically a function number in the card.
P1	Parameter 1	1	Instruction parameter 1
P2	Parameter 2	1	Instruction parameter 2
Ex	Extended indicator	missing or 1	With value 00 <sub>hex</sub> indicates that the command data has extended format. Look chapter <a href="#">7.5 Extended APDU</a> .
Lc	Length	variable 1 or 2	Number of bytes present in the data field of the command

**Table 7-2 C-APDU contents**

Code	Name	Length	Description
Data	Data	variable, equal to Lc	String of bytes sent in the data field of the command
Le	Length	variable 1 or 2	Maximum number of bytes expected in the data field of the response to the command



Keep in mind that by using T1 protocol either Le or data field has to be present always. When there is no specific value for Le or data field while using T1, then Le field must be set to value 00<sub>hex</sub>.

**Table 7-3 R-APDU structure**

Body	Trailer	
Data	SW1	SW2

**Table 7-4 R-APDU contents**

Code	Name	Length	Description
Data	Data	variable, equal to Le if was present in C-APDU	Sequence of bytes received in the data field of the response (Optional field)
SW1	Status byte 1	1	Command processing status
SW2	Status byte 2	1	Command processing qualifier

## 7.1. Card possible response in case of protocol T0

When using protocol T0 and sending a C-APDU that should return data, the card responds with R-APDU that informs the host how many bytes are waiting to be read. The sequence of operations in given case is described in following example:

- 1) The card responds to the C-APDU with trailer 61XX<sub>hex</sub> as a positive response. XX<sub>hex</sub> in the response indicates how many bytes of data are waiting to be read from the card:

SW1	SW2
61 <sub>hex</sub>	XX <sub>hex</sub>

- 2) In order to read the given bytes, the following GET RESPONSE command must be sent to the card:

CLA	INS	P1	P2	Le
00 <sub>hex</sub>	C0 <sub>hex</sub>	00 <sub>hex</sub>	00 <sub>hex</sub>	XX <sub>hex</sub>

If more bytes are waiting to be read from the card, the card keeps responding 61XX<sub>hex</sub> after every GET RESPONSE command execution as long there will be none waiting to be read.



3) The card responds:

Data	SW1	SW2
XX <sub>hex</sub> bytes of data	90 <sub>hex</sub>	00 <sub>hex</sub>

For T0 there is also possible a case when card responds with status code which informs to reissue the same APDU command with Le byte set as marked in status. In given case card responds as follows where XX<sub>hex</sub> marks Le byte value that should be used when reissuing the command:

SW1	SW2
6C <sub>hex</sub>	XX <sub>hex</sub>

After reissuing the APDU command the card responds as normally.

## 7.2. Command APDU

Card application APDU commands are derived from ISO 7816-4 but do not implement all given specification functionalities. Implemented is minimal of required functionalities for EstEID PKI operations. Current chapter gives detailed usage information of the implemented functions. All implemented APDU commands are listed in the following table.

Table 7-5 Card application implemented APDU commands		
Command name	INS	Description
<u>SELECT FILE</u>	A4 <sub>hex</sub>	To change pointer for currently selected file.
<u>READ RECORD</u>	B2 <sub>hex</sub>	Reads data from Linear EF.
<u>READ BINARY</u>	B0 <sub>hex</sub>	Reads data from Transparent EF.
<u>GET RESPONSE</u>	C0 <sub>hex</sub>	Receive data from card in case of status 61XX <sub>hex</sub> .
<u>GET DATA</u>	CA <sub>hex</sub>	Read data related to application or chip. <ul style="list-style-type: none"> <li>▪ Application version</li> <li>▪ CPLC</li> <li>▪ Available memory on chip</li> </ul>
<u>GET CHALLENGE</u>	84 <sub>hex</sub>	Generates and returns random number for authentication purposes.
<u>VERIFY</u>	20 <sub>hex</sub>	To verify the presented user PIN1/PIN2/PUK code against the stored reference values.
<u>CHANGE REFERENCE DATA</u>	24 <sub>hex</sub>	To change PIN1/PIN2/PUK code values on the card.
<u>RESET RETRY COUNTER</u>	2C <sub>hex</sub>	To reset PIN1 or PIN2 retry counters.
<u>MANAGE SECURITY ENVIRONMENT</u>	22 <sub>hex</sub>	Sets currently active key environment for cryptographic operations.





Table 7-5 Card application implemented APDU commands		
Command name	INS	Description
INTERNAL AUTHENTICATE	88 <sub>hex</sub>	To authenticate card by the host side
MUTUAL AUTHENTICATE	82 <sub>hex</sub>	To authenticate host for card management operations.
PERFORM SECURITY OPERATION <ul style="list-style-type: none"> <li>HASH</li> <li>DECIPHER</li> <li>COMPUTE DIGITAL SIGNATURE</li> </ul>	2A <sub>hex</sub>	Functions to perform various cryptographic operations with card application RSA key pair private keys.
EstEID card application authority operations.		
REPLACE PINS (SECURE)	05 <sub>hex</sub>	To replace PIN/PUK codes for cardholder.
GENERATE KEY (SECURE)	06 <sub>hex</sub>	To generate new RSA key pair for cardholder.
REPLACE CERTIFICATE (SECURE)	07 <sub>hex</sub>	To replace cardholder certificate.

### 7.2.1. SELECT FILE

CLA	INS	P1	P2	Lc	Data	Le	Command description
00 <sub>hex</sub>	A4 <sub>hex</sub>	0X <sub>hex</sub>	00 <sub>hex</sub>	Empty or 2. Described in P1 table below.	according to P1 field	empty or 00 <sub>hex</sub>	Response includes FCI (FCP+FMD)
		0X <sub>hex</sub>	04 <sub>hex</sub>				Response includes FCP
		0X <sub>hex</sub>	08 <sub>hex</sub>				Response includes FMD
		0X <sub>hex</sub>	0C <sub>hex</sub>				Response includes only OK status 0x9000 if successful

The SELECT FILE command is used to change the logical pointer of currently selected file to perform operations on. The file identification can be provided by file identifier (FID) on 2 bytes.

Selected file logical pointer changing method is defined in P1 field. These methods are provided in following table:

Value of X in P1	Lc	Data	Description
0	empty or	empty or	Select application MF
	2	3F00 <sub>hex</sub>	
1	2	FID	Select DF with file identifier declared in Data



Value of X in P1	Lc	Data	Description
2	2	FID	Select EF with file identifier declared in Data
3	empty	empty	Select parent DF of currently selected DF

Card application can answer to given command with R-APDU described in following table.

Data	SW1	SW2	Description
empty or data containing FCI, FCP or FMD	90 <sub>hex</sub>	00 <sub>hex</sub>	Successfully changed the file pointer (and returned Data)
	64 <sub>hex</sub>	09 <sub>hex</sub>	Could not generate FCP for selectable DF.
	67 <sub>hex</sub>	00 <sub>hex</sub>	Invalid length of data for provided P1 value
	6A <sub>hex</sub>	80 <sub>hex</sub>	Invalid FID for MF
	6A <sub>hex</sub>	82 <sub>hex</sub>	File not found for provided FID
	6A <sub>hex</sub>	86 <sub>hex</sub>	Possible reasons: <ul style="list-style-type: none"> <li>invalid P1 value</li> <li>invalid P2 value</li> </ul>

### 7.2.2. READ RECORD

CLA	INS	P1	P2	Lc	Data	Le
00 <sub>hex</sub>	B2 <sub>hex</sub>	Record no. of the record to be read	04 <sub>hex</sub>	empty	empty	00 <sub>hex</sub> or exactly the length of the record

The READ RECORD command is used to read data records from Linear EF. Linear EF is structured file containing records.

Card application answers to given command with R-APDU described in following table.

Data	SW1	SW2	Description
Data with length of record or provided by Le value	90 <sub>hex</sub>	00 <sub>hex</sub>	Successfully read the record
	62 <sub>hex</sub>	82 <sub>hex</sub>	Record's value is shorter than provided in Le
	67 <sub>hex</sub>	00 <sub>hex</sub>	Record's value is longer than provided in Le
	69 <sub>hex</sub>	81 <sub>hex</sub>	Trying to read record from non-Linear EF
	69 <sub>hex</sub>	86 <sub>hex</sub>	Missing selection pointer for EF

Data	SW1	SW2	Description
	6A <sub>hex</sub>	83 <sub>hex</sub>	The Requested record is not found.
	6A <sub>hex</sub>	86 <sub>hex</sub>	P2 value is not 04 <sub>hex</sub>

### 7.2.3. READ BINARY

CLA	INS	P1    P2	Lc	Data	Le
00 <sub>hex</sub>	B0 <sub>hex</sub>	XXXX <sub>hex</sub> Offset to start a reading from the file	empty	empty	number of bytes to read

The READ BINARY command is used to read binary data from Transparent EF.



There is no need to read whole data from the file with multiple READ BINARY commands with different file reading offset for each command. READ BINARY command supports extended length response. See chapter 7.5 Extended APDU.

Card application answer to given command with R-APDU described in following table.

Data	SW1	SW2	Description
Binary data with length of data or provided by Le	90 <sub>hex</sub>	00 <sub>hex</sub>	Successfully read the binary data from EF
Binary data with shorter length than requested by Le value	62 <sub>hex</sub>	82 <sub>hex</sub>	Returned file contents but warning, that data that was read was shorter than requested.
	69 <sub>hex</sub>	81 <sub>hex</sub>	Trying to read binary data from non-Transparent EF
	69 <sub>hex</sub>	86 <sub>hex</sub>	Missing selection pointer for EF
	6B <sub>hex</sub>	00 <sub>hex</sub>	Offset is bigger than file actual size

### 7.2.4. GET RESPONSE

CLA	INS	P1	P2	Lc	Data	Le
00 <sub>hex</sub>	C0 <sub>hex</sub>	00 <sub>hex</sub>	00 <sub>hex</sub>	empty	empty	XX <sub>hex</sub>

The GET RESPONSE command is used for protocol T0 to get data from the card, which is sent by the card implicitly. Before given command can be executed, the card must send status 61XX<sub>hex</sub>, where XX marks the length of a data available for returning from the card. For GET RESPONSE command the same value of XX must be used as received in status.

Card application answer to given command with R-APDU described in following table.

Data	SW1	SW2	Description
Data with length as provided in Le	90 <sub>hex</sub>	00 <sub>hex</sub>	Successful operation
	61 <sub>hex</sub>	XX <sub>hex</sub>	Additional data waiting to be returned from the card.
	--	--	No other errors than defined in chapter 7.3 Error response APDU messages.

### 7.2.5. GET DATA

CLA	INS	P1	P2	Lc	Data	Le
00 <sub>hex</sub>	CA <sub>hex</sub>	XX <sub>hex</sub>	00 <sub>hex</sub>	empty	empty	00 <sub>hex</sub>

The GET DATA command is used to get various information of the EstEID application and card itself. The information that the card should return is defined in P1 field. Possible P1 values and result descriptions are specified in the following table:

P1	Description for data returned by card
01 <sub>hex</sub>	EstEID application version.
02 <sub>hex</sub>	CPLC data for chip.
03 <sub>hex</sub>	Current free memory on the card.

Card application answer to given command with R-APDU described in following table.

Data	SW1	SW2	Description
XXXX <sub>hex</sub>	90 <sub>hex</sub>	00 <sub>hex</sub>	P1 = 01 <sub>hex</sub> – EstEID application version as BCD.
42 bytes of CPLC data.			P1 = 02 <sub>hex</sub> – CPLC data for chip.
XXXX <sub>hex</sub>			P1 = 03 <sub>hex</sub> Free transient memory that is freed on application deselecting or reset.
YYYY <sub>hex</sub>			Free transient memory that is freed on application reset.
ZZZZ <sub>hex</sub>			Free persistent type memory.
	6A <sub>hex</sub>	86 <sub>hex</sub>	Invalid P1 value



If there is more free memory available than FFFF<sub>hex</sub> then given memory value will be returned as FFFF<sub>hex</sub>.



### 7.2.6. GET CHALLENGE

CLA	INS	P1	P2	Lc	Data	Le
00 <sub>hex</sub>	84 <sub>hex</sub>	00 <sub>hex</sub>	00 <sub>hex</sub>	empty	empty	00 <sub>hex</sub> or 08 <sub>hex</sub> or XX <sub>hex</sub>

The GET CHALLENGE command is used to receive a challenge (e.g. random number) for use in a security related procedure.

Le field defines the length for data that should be generated in the card. If Le field is empty or has value 00<sub>hex</sub> then the length of random is considered to be 08<sub>hex</sub>. Only Le field with value 08<sub>hex</sub> result is stored for further internal operations. Random numbers generated with other length are only for off card usage.

Card application answer to given command with R-APDU described in following table.

Data	SW1	SW2	Description
Data with length provided by Le.	90 <sub>hex</sub>	00 <sub>hex</sub>	Random data.
	6A <sub>hex</sub>	86 <sub>hex</sub>	Invalid P1 or P2 value.

### 7.2.7. VERIFY

CLA	INS	P1	P2	Lc	Data	Le	Description
00 <sub>hex</sub>	20 <sub>hex</sub>	00 <sub>hex</sub>	01 <sub>hex</sub>	04 <sub>hex</sub> ...0C <sub>hex</sub>	PIN1	empty	PIN and PUK codes verification operations
			02 <sub>hex</sub>	05 <sub>hex</sub> ...0C <sub>hex</sub>	PIN2		
			00 <sub>hex</sub>	08 <sub>hex</sub> ...0C <sub>hex</sub>	PUK		

The VERIFY command is used to authenticate cardholder through PIN1, PIN2 or PUK code.

Upper and lower limits for the lengths of PIN1, PIN2 and PUK are marked in Lc field. Default verification data length for given verification method is the minimum marked in Lc field. Other lengths of verification data can be used after successful operation of command CHANGE REFERENCE DATA.



PIN1, PIN2 and PUK codes must be provided in communication as ASCII character numbers.



Unsuccessful operation of given command results in decrementing the corresponding PIN/PUK code retry counter.

Card application answer to given command with R-APDU described in following table.

Data	SW1	SW2	Description
empty	90 <sub>hex</sub>	00 <sub>hex</sub>	Successful verification.
	63 <sub>hex</sub>	CX <sub>hex</sub>	Verification failed. X in SW2 marks remaining tries for verification.
	67 <sub>hex</sub>	00 <sub>hex</sub>	Verification data can't be empty.
	69 <sub>hex</sub>	83 <sub>hex</sub>	Verification method blocked.



Data	SW1	SW2	Description
	6A <sub>hex</sub>	80 <sub>hex</sub>	PIN/PUK code value is out of the expected limits range.
	6A <sub>hex</sub>	86 <sub>hex</sub>	Invalid P1 or P2 value.

### 7.2.8. CHANGE REFERENCE DATA

CLA	INS	P1	P2	Lc	Data	Le	Ref. data
00 <sub>hex</sub>	24 <sub>hex</sub>	00 <sub>hex</sub>	01 <sub>hex</sub>	old length + new length	old PIN1    new PIN1	empty	PIN1
			02 <sub>hex</sub>	old length + new length	old PIN2    new PIN2		PIN2
			00 <sub>hex</sub>	old length + new length	old PUK    new PUK		PUK

The CHANGE REFERENCE DATA command is used to replace PIN1, PIN2 or PUK code. To change PIN1/PIN2/PUK code it is needed to know the currently active code. It is allowed to assign only new PIN1/PIN2/PUK which is different from the current one.



PIN1, PIN2 and PUK codes must be provided in communication as ASCII character numbers.



Unsuccessful operation of given command results in decrementing the corresponding PIN/PUK code retry counter.

Card application answer to given command with R-APDU described in following table.

Data	SW1	SW2	Description
empty	90 <sub>hex</sub>	00 <sub>hex</sub>	Reference data successfully changed.
	63 <sub>hex</sub>	CX <sub>hex</sub>	Verification failed. X in SW2 marks remaining tries for verification.
	67 <sub>hex</sub>	00 <sub>hex</sub>	Verification data can't be empty.
	69 <sub>hex</sub>	83 <sub>hex</sub>	Verification method blocked.
	69 <sub>hex</sub>	85 <sub>hex</sub>	If length is invalid
	6A <sub>hex</sub>	80 <sub>hex</sub>	<ul style="list-style-type: none"> <li>Old and new PIN/PUK are equal.</li> <li>New PIN/PUK is out of the expected limits range.</li> </ul>
	6A <sub>hex</sub>	86 <sub>hex</sub>	Invalid P1 or P2 value.



### 7.2.9. RESET RETRY COUNTER

CLA	INS	P1	P2	Lc	Data	Le	Description
00 <sub>hex</sub>	2C <sub>hex</sub>	00 <sub>hex</sub>	0X <sub>hex</sub>	PUK + PIN1/PIN 2 length	PUK new PIN1/PIN2	empty	Verify PUK and assign new PIN1/PIN2.
		03 <sub>hex</sub>	0X <sub>hex</sub>	empty	empty	00 <sub>hex</sub>	Reset PIN1/PIN2. PUK pre- verified.
		0X <sub>hex</sub>	01 <sub>hex</sub>	Defined by P1		empty	Operation with PIN1.
		0X <sub>hex</sub>	02 <sub>hex</sub>			y	Operation with PIN2.

The RESET RETRY COUNTER command is used to replace reset or unblock PIN1 or PIN2 code.

To use P1 with value 03<sub>hex</sub> it is needed to have command VERIFY PUK used. For operation P1 with value 00<sub>hex</sub> for verification it is needed to provide PUK code as well with the new PIN1/PIN2 code.

The command cannot be used for PIN1/PIN2 which is in blocked state.



PIN1, PIN2 and PUK codes should be provided in communication as ASCII character numbers.



Unsuccessful operation of given command can result in decrementing PUK code retry counter if P1 as value 00<sub>hex</sub> used.

Card application answer to given command with R-APDU described in following table.

Data	SW1	SW2	Description
empty	90 <sub>hex</sub>	00 <sub>hex</sub>	Successful operation
	63 <sub>hex</sub>	CX <sub>hex</sub>	Verification failed. X in SW2 marks remaining tries for PUK verification.
	69 <sub>hex</sub>	82 <sub>hex</sub>	PUK not pre-verified
	69 <sub>hex</sub>	83 <sub>hex</sub>	Verification method blocked.
	69 <sub>hex</sub>	85 <sub>hex</sub>	PIN trying to reset, is not blocked.
	6A <sub>hex</sub>	86 <sub>hex</sub>	Invalid P1 or P2 value.



## 7.2.10. MANAGE SECURITY ENVIRONMENT

CLA	INS	P1	P2	Lc	Data	Le	Description
00 <sub>hex</sub>	22 <sub>hex</sub>	F3 <sub>hex</sub>	01 <sub>hex</sub>	empty	empty	00 <sub>hex</sub>	Set security environment for signing and authentication operations.
			06 <sub>hex</sub>				Set security environment for deciphering operation.
		41 <sub>hex</sub>	A4 <sub>hex</sub>	02 <sub>hex</sub>	8300 <sub>hex</sub>	empty	Reset key references to active ones.
			B4 <sub>hex</sub> B6 <sub>hex</sub> B8 <sub>hex</sub>	05 <sub>hex</sub>	830380 <sub>hex</sub> XXXX <sub>hex</sub>		Set key reference to specific key by providing key reference in the position of XXXX.

The MANAGE SECURITY ENVIRONMENT command is used to change the currently active pointers to keys for security operations.

Card application answer to given command with R-APDU described in following table.

Data	SW1	SW2	Description
empty	90 <sub>hex</sub>	00 <sub>hex</sub>	Successful operation
	67 <sub>hex</sub>	00 <sub>hex</sub>	Invalid length of data for provided command parameters.
	69 <sub>hex</sub>	83 <sub>hex</sub>	Verification method blocked.
	6A <sub>hex</sub>	86 <sub>hex</sub>	Invalid P1 or P2 value.
	6A <sub>hex</sub>	80 <sub>hex</sub>	Incorrect data (invalid length).

## 7.2.11. INTERNAL AUTHENTICATE

CLA	INS	P1	P2	Lc	Data	Le
00 <sub>hex</sub>	88 <sub>hex</sub>	00 <sub>hex</sub>	00 <sub>hex</sub>	Token length	Authentication token	Empty

The INTERNAL AUTHENTICATE command is used to authenticate the cardholder by the host side. Data field in C-APDU must contain token that will be encrypted with private key stored in the card. Challenge can be verified by using public key of the same key pair.

Card application answer to given command with R-APDU described in following table.

Data	SW1	SW2	Description
RSA authentication private key encrypted TLS challenge which is formatted according to PKCS#1 ver. 1.5 block type 1.	90 <sub>hex</sub>	00 <sub>hex</sub>	Successful. Data field contains encrypted challenge.
	69 <sub>hex</sub>	00 <sub>hex</sub>	Security environment is not set to Authenticate





Data	SW1	SW2	Description
	69 <sub>hex</sub>	82 <sub>hex</sub>	PIN1 is not validated.
	6A <sub>hex</sub>	86 <sub>hex</sub>	Invalid P1 or P2 value.
	6A <sub>hex</sub>	80 <sub>hex</sub>	Token has invalid length. Has to fit into RSA key length.

### 7.2.12. MUTUAL AUTHENTICATE

CLA	INS	P1	P2	Lc	Data	Le	For use of
00 <sub>hex</sub>	82 <sub>hex</sub>	00 <sub>hex</sub>	01 <sub>hex</sub>	30 <sub>hex</sub>	CMK encrypted RND.IFD    RND.ICC    K.IFD	empty or 00 <sub>hex</sub> or 30 <sub>hex</sub>	CMK_PIN
			02 <sub>hex</sub>				CMK_CERT
			03 <sub>hex</sub>				CMK_KEY

The MUTUAL AUTHENTICATION command is used for host authentication. After successful operation of given command card management commands can be used over secure encrypted channel.

The whole process of mutual authentication is described in chapter 4.1.1 Mutual Authentication.

Card application answer to given command with R-APDU described in following table.

Data	SW1	SW2	Description
Corresponding CMK encrypted RND.ICC    RND.IFD    K.ICC	90 <sub>hex</sub>	00 <sub>hex</sub>	Successful operation. Data field contains encrypted card and host challenges and card session key.
	63 <sub>hex</sub>	CF <sub>hex</sub>	Mutual authentication failed.
	64 <sub>hex</sub>	00 <sub>hex</sub>	Incorrect P2 value. No such CMK
	67 <sub>hex</sub>	00 <sub>hex</sub>	Data has length different to 30 <sub>hex</sub>
	6A <sub>hex</sub>	86 <sub>hex</sub>	Invalid P1 value.

### 7.2.13. PERFORM SECURITY OPERATION

CLA	INS	P1	P2	Lc	Data	Le
0X <sub>hex</sub>	2A <sub>hex</sub>	XX	XX	XX	XX	empty

The PERFORM SECURITY OPERATION command is for three cryptographic algorithms:

- HASH – calculates bit hash from the data transferred by the command.
- DECIPHER – decrypts cryptogram which is transferred by the command.
- COMPUTE DIGITAL SIGNATURE – computes digital signature for the data transferred by the command.

### 7.2.13.1.HASH



This feature is deprecated for EstEID version v3.5.7 and above.

CLA	INS	P1	P2	Lc	Data	Le	Description
00 <sub>hex</sub>	2A <sub>hex</sub>	90 <sub>hex</sub>	A0 <sub>hex</sub>	Data length	Data hashing	empty	Last data block.
10 <sub>hex</sub>							Chain data block.

The HASH command is used to calculate unique data value on provided data. Algorithm used for hashing is SHA1.

Card application answer to given command with R-APDU described in following table.

Data	SW1	SW2	Description
Generated hash for provided data.	90 <sub>hex</sub>	00 <sub>hex</sub>	Successful operation.
	--	--	No other errors than defined in chapter 7.3 Error response APDU messages.

### 7.2.13.2.DECIPHER

CLA	INS	P1	P2	Lc	Data (Data for deciphering)	Le	Description
00 <sub>hex</sub>	2A <sub>hex</sub>	80 <sub>hex</sub>	86 <sub>hex</sub>	Data length		empty	Last data block.
10 <sub>hex</sub>					00 <sub>hex</sub>    cryptogram		Chain data block.

The DECIPHER command is used to decipher data provided by the command. Data has to be formatted according to PKCS#1 ver. 1.5 block type 2 with the respective public key. Given operation can be performed only with private authentication keys.



Data transmitted to the card application for deciphering must be pre-padded with 00<sub>hex</sub> byte which indicates that the data under the cryptogram is formatted according to PKCS#1 ver. 1.5 block type 2:

00<sub>hex</sub> || cryptogram



Needs PIN1 to be pre-verified.



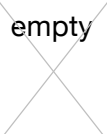
DECIPHER command supports chaining and extended C-APDU for data transmitting. For extended APDU see chapter 7.5 Extended APDU and for chaining see chapter 7.4 Message chaining.

Card application answer to given command with R-APDU described in following table.

Data	SW1	SW2	Description
Deciphered data.	90 <sub>hex</sub>	00 <sub>hex</sub>	Successful deciphering.

Data	SW1	SW2	Description
	69 <sub>hex</sub>	00 <sub>hex</sub>	Security environment is not set to Decipher.
	69 <sub>hex</sub>	82 <sub>hex</sub>	PIN1 is not validated.

### 7.2.13.3.COMPUTE DIGITAL SIGNATURE

CLA	INS	P1	P2	Lc	Data	Le	Description
00 <sub>hex</sub>	2A <sub>hex</sub>	9E <sub>hex</sub>	9A <sub>hex</sub>	Data length	Data for signature		Data signing
				00 <sub>hex</sub>	empty		Hash signing

The COMPUTE DIGITAL SIGNATURE command is used to compute unique signature for data or for hash generated previous to this command. For signature it is used in card private key of RSA key pair.




Needs PIN2 to be pre-verified.

Card application answer to given command with R-APDU described in following table.

Data	SW1	SW2	Description
Computed signature.	90 <sub>hex</sub>	00 <sub>hex</sub>	Successful operation.
	69 <sub>hex</sub>	00 <sub>hex</sub>	Security environment is not set to Digital Signature.
	69 <sub>hex</sub>	82 <sub>hex</sub>	PIN2 is not validated.
	6A <sub>hex</sub>	88 <sub>hex</sub>	Missing hash that has to be generated prior to this command with HASH command.

### 7.2.14. REPLACE PINS (SECURE)


CLA	INS	P1	P2	Lc	Data (PIN1, PIN2 and PUK are as ASCII)	Le
0C <sub>hex</sub>	05 <sub>hex</sub>	00 <sub>hex</sub>	00 <sub>hex</sub>	11 <sub>hex</sub>	PIN1    PIN2    PUK	

The REPLACE PINS command is used to replace current PINs and PUK codes with new ones by EstEID card authority.



Given APDU command requires secure communication channel for processing, as described in chapter 4.1 [Secure channel communication](#).

Card application answer to given command with R-APDU described in following table.

Data	SW1	SW2	Description
	90 <sub>hex</sub>	00 <sub>hex</sub>	Successful operation.



Data	SW1	SW2	Description
	6A <sub>hex</sub>	86 <sub>hex</sub>	Invalid P1 or P2 value.
	69 <sub>hex</sub>	86 <sub>hex</sub>	Wrong Mutual Authentication CMK key used for current command.
	67 <sub>hex</sub>	00 <sub>hex</sub>	Missing hash that has to be generated prior to this command with HASH command.

### 7.2.15. GENERATE KEY (SECURE)

CLA	INS	P1	P2	Lc	Data	Le	Description
0C <sub>hex</sub>	06 <sub>hex</sub>	01 <sub>hex</sub>	01 <sub>hex</sub>	empty	empty	empty	Authentication key nr 1
			02 <sub>hex</sub>			or 00 <sub>hex</sub>	Signature key nr 1
		02 <sub>hex</sub>	01 <sub>hex</sub>			or 010F <sub>hex</sub>	Authentication key nr 2
			02 <sub>hex</sub>			as extended	Signature key nr 2

The GENERATE KEY command is used to generate new RSA key pairs by EstEID card authority.



Given APDU command requires secure communication channel for processing, as described in chapter [4.1 Secure channel communication](#).



Accessing given command requires as well the acceptance from the cardholder by verifying PIN1 code with command [VERIFY](#).

Card application answer to given command with R-APDU described in following table.

Data (271 <sub>dec</sub> bytes)	SW1	SW2	Description
7F49 <sub>hex</sub>    82010A <sub>hex</sub>    81 <sub>hex</sub>    820100 <sub>hex</sub>    public key    82 <sub>hex</sub>    04 <sub>hex</sub>    exponent	90 <sub>hex</sub>	00 <sub>hex</sub>	Successful operation. Data field containing TLV data for public key of generated RSA key pair.
	6A <sub>hex</sub>	86 <sub>hex</sub>	Invalid P1 or P2 value.
	69 <sub>hex</sub>	86 <sub>hex</sub>	Wrong Mutual Authentication CMK key used for current command.
	67 <sub>hex</sub>	00 <sub>hex</sub>	Missing hash that has to be generated prior to this command with HASH command.



For detailed information of the template look ISO 7816-8.

### 7.2.16. REPLACE CERTIFICATE (SECURE)

CLA	INS	P1 8bit	P1 7-1bit	P2	Lc	Data	Le	Description
0C <sub>hex</sub>	06 <sub>hex</sub>	0 <sub>bit</sub>	XX <sub>hex</sub> (00-7F <sub>hex</sub> )	XX <sub>hex</sub>	Data length	Certificate data	empty	Authentication certificate
		1 <sub>bit</sub>						Signature certificate

The REPLACE CERTIFICATE command is used to replace cardholder authentication or signature certificate by EstEID card authority.

The maximum length of data for the certificate can be is 600<sub>hex</sub> bytes. The certificate data must fit into this length and have padding according to ISO 9797-1 padding method 2.

The certificate file must be written to card as multiple chunks. Each following chunk must be send to the card by using offset of the last sent data.



Given APDU command requires secure communication channel for processing, as described in chapter 4.1 Secure channel communication.



Accessing given command requires as well the acceptance from the cardholder by verifying PIN1 code with command VERIFY.

Card application answer to given command with R-APDU described in following table.

Data	SW1	SW2	Description
empty	90 <sub>hex</sub>	00 <sub>hex</sub>	Successful operation.
	69 <sub>hex</sub>	86 <sub>hex</sub>	Wrong Mutual Authentication CMK key used for current command or PIN1 is not successfully validated.
	6A <sub>hex</sub>	86 <sub>hex</sub>	Invalid P1 or P2 value.

### 7.3. Error response APDU messages

Error codes provided in the following table can be returned by any of C-APDUs. These errors are not the result of the usual command processing.

Table 7-6 APDU error status codes		
SW1	SW2	Definition
68 <sub>hex</sub>	84 <sub>hex</sub>	Command chaining not supported.
6D <sub>hex</sub>	00 <sub>hex</sub>	Command instruction not supported.
6E <sub>hex</sub>	00 <sub>hex</sub>	Command class not supported.
6F <sub>hex</sub>	00 <sub>hex</sub>	No precise diagnosis.
6F <sub>hex</sub>	66 <sub>hex</sub>	Internal inconsistency.

## 7.4. Message chaining

This chapter explains the basis of APDU message chaining in case of larger data to be transmitted.

JavaCard framework 2.2.2 and above support extended length APDU messages. Still TPDU T0 protocol does not support extended length APDU processing. Thus the data to be transmitted between the host and the card which doesn't fit into the usual length of APDU messages must be transmitted as chained in the case of T0 usage.

EstEID card application does not respond with data longer than maximum of standard APDU response messages. Thus there is no need for message chaining in given case.

The maximum length of data that can be sent by normal APDU is 255<sub>dec</sub> bytes. If there is more data to transfer to the card than the maximum length then it is needed to split the data into as many blocks it could be delivered to the card.

The execution of the operation takes place when the last block is received by the card. Chained blocks and last block of data are distinguished by the command class. The command class bit which determines that the command is a part of the chained command sequence is 10<sub>hex</sub>. The last block of the sequence must have the chaining bit set off in command class.

For getting a better overview, following example where it is needed to send 700<sub>dec</sub> bytes of data to the card for card internal processing.

### 1) First C-APDU:

CLA	INS	P1	P2	Lc	Data	Le
10 <sub>hex</sub>	XX	XX	XX	255 <sub>dec</sub>	First block of 255 <sub>dec</sub> bytes of 700 <sub>dec</sub> bytes	XX

Data	SW1	SW2	Description
empty	90 <sub>hex</sub>	00 <sub>hex</sub>	Block received. Waiting for another one.

### 2) Second C-APDU:

CLA	INS	P1	P2	Lc	Data	Le
10 <sub>hex</sub>	XX	XX	XX	255 <sub>dec</sub>	Second block of 255 <sub>dec</sub> bytes of 700 <sub>dec</sub> bytes	XX

Data	SW1	SW2	Description
empty	90 <sub>hex</sub>	00 <sub>hex</sub>	Block received. Waiting for another one.

### 3) Third and last C-APDU:

CLA	INS	P1	P2	Lc	Data	Le
00 <sub>hex</sub>	XX	XX	XX	190 <sub>dec</sub>	Last block of 190 <sub>dec</sub> bytes of 700 <sub>dec</sub> bytes	XX

Data	SW1	SW2	Description
Operation result data	90 <sub>hex</sub>	00 <sub>hex</sub>	Last block received. Operation successful and result data returned.

## 7.5. Extended APDU

As mentioned in previous chapter JavaCard framework 2.2.2 and above support extended length APDU messages. If data for the command exceeds the maximum of normal C-APDU, which is 255<sub>dec</sub> bytes, the data can be sent as extended. Maximum length for extended length data that can be transmitted is 65536<sub>dec</sub> bytes – the actual size of short data type.



Extended APDU can only be used with protocol T1.

In extended C-APDU in Lc and Le fields have length of 2 bytes. Extended C-APDU has Le field always present. Lc field is optional. C-APDU body must always be pre-padded with 00<sub>hex</sub> which is the indicator of extended APDU.

- If Lc and data fields are present:

CLA	INS	P1	P2	Ex. APDU indicator	Lc	Data	Le
XX	XX	XX	XX	00 <sub>hex</sub>	XXXX <sub>hex</sub>	Extended data	XXXX <sub>hex</sub>

- If Lc and data fields are absent:

CLA	INS	P1	P2	Ex. APDU indicator	Lc	Data	Le
XX	XX	XX	XX	00 <sub>hex</sub>	empty	empty	XXXX <sub>hex</sub>

Extended length R-APDU does not have differences compared to normal one. Just the data that is returned by R-APDU is exceeding the length of 256<sub>dec</sub> bytes.



## Abbreviations

Table of Abbreviations	
Abbreviation	Definition
AID	Application identifier – sequence of bytes (5 <sub>dec</sub> -16 <sub>dec</sub> ) to identify the application on the card.
ANSI	American National Standards Institute
APDU	Application Protocol Data Unit - The application protocol data unit of the chip.
ASCII	American Standard Code for Information Interchange - The standard 7-bit code table to present digitally the English alphabet and other keyboard symbols.
ASN.1 BER	Abstract Syntax Notation One Basic Encoding Rules. Often called as Tag Length Value (TLV) formatting rules.
ASN.1 DER	Abstract Syntax Notation One Distinguished Encoding Rules.
HEX	The symbol for the hexadecimal numeral system.
BCD	The presentation of numbers in a way that the first 4 and last 4 bits of each byte could be viewed as separate digits ranging 0 through 9 in the hexadecimal numeral system.
CPLC	Card Production Life Cycle – Data containing information about chip fabricator, operating system, chip serial number, personalisers, personalising equipment, personalising dates, etc.
BIN	The symbol for the binary number system.
DEC	The symbol for the decimal number system.
ICC	Integrated Circuit Card
IFD	Interface Device
K	Key
KID	The key identifier byte in key reference data.
KST	Key search type
KV	The key version byte in key reference data.
LSB	The least significant bit.
MSB	The most significant bit.
FID	The file identifier.
FCI	File Control Information. Data FCP+FMD. (ISO 7816-4)
FCP	File Control Parameters. The given parameters include a list of logical, structural and security attributes. (ISO 7816-4)
FMD	File Management Data (ISO 7816-4)
POS	Point of sale





Table of Abbreviations	
Abbreviation	Definition
RND	Random
TLV	Tag Length Value – Data formatting method.

## Terms

Table of Terms	
Term	Definition
Authentication	The procedure for confirming the origin of somebody or something.
Authorisation	The procedure during which it is established whether the given person has the rights for the particular operation.
Digital signing	The procedure that results as an unique verifiable data.
Hash	A unique set of bits corresponding to a specific set of data.
PIN – code	Personal Identification Number - A code consisting of letters and/or digits used to authenticate the user identity.
Verification	The procedure for verifying the data validity.

## References

Kaliski, B., "PKCS #1: RSA Encryption Version 1.5", RFC 2313, March 1998.

ISO/IEC 7816-3 (2006): "Identification cards - Integrated circuit cards - Part 3: Cards with contacts — Electrical interface and transmission protocols".

ISO/IEC 7816-4 (2013): "Identification cards - Integrated circuit cards - Part 4: Organization, security and commands for interchange".

ISO/IEC 7816-8 (2014): "Identification cards - Integrated circuit cards - Part 8: Commands for security operations".

ISO/IEC 18033-3 (2010): "Information technology - Security techniques - Encryption algorithms - Part 3: Block ciphers".

ISO/IEC 9797-1 (1999): "Information technology - Security techniques - Message Authentication Codes (MACs) - Part 1: Mechanisms using a block cipher".

Java Card™ Specifications Version 2.2.2, March 2006



## List of Tables

Table for Document Version History .....	1
Table 1-1The functions of EstEID security chip objects in the card application ..	11
Table 2-1Personal Data file contents .....	13
Table 2-2PIN1, PIN2 and PUK .....	14
Table 2-3EF FID 0016 <sub>hex</sub> contents .....	18
Table 2-4EF FID 0013 <sub>hex</sub> key records and references of secret keys .....	23
Table 2-5EF FID 0013 <sub>hex</sub> key record description .....	23
Table 2-6EF FID 0033 <sub>hex</sub> key record description .....	23
Table 2-7Card Production Life Cycle data .....	28
Table 4-1C-APDUs supporting secure channel .....	37
Table 6-1Card application constant values .....	45
Table 7-1C-APDU structure .....	46
Table 7-2C-APDU contents .....	46
Table 7-3R-APDU structure .....	47
Table 7-4R-APDU contents .....	47
Table 7-5Card application implemented APDU commands .....	48
Table 7-6APDU error status codes .....	61
Table of Abbreviations .....	64
Table of Terms .....	65

## List of Pictures

Figure 1-1 EstEID filesystem diagram .....	10
Figure 2-1 Card application objects .....	12
Figure 4-1 DES and 3DES CBC mode encryption .....	39
Figure 4-2 MAC signature calculation and 3DES CBC mode decryption .....	40
Figure 7-1 APDU master-slave communication .....	45





## APPENDIX

This appendix contains real life card application operations logs, which should give a better overview of the commands. Following operations are performed with test environment card application using transmission protocol T1 with Le always present.

Master PIN/PUK codes and CMK keys used in following operations are the same as mentioned in chapter 2.2.1 Verify PIN1, PIN2 or PUK code and 2.5 Card application management keys: CMK\_PIN, CMK\_CERT & CMK\_.

### Reset the chip with EstEID card application installed on

Chip responds with ATR.

```
<< 3B FE 18 00 00 80 31 FE 45 45 73 74 45 49 44 20 76 65 72 20 31 2E 30 A8
TS : 3B Direct logic
TO : FE K = 14 byte [historical characters]
TA1 : 18 Fi/f = 372/ 5 [clock rate conversion factor / max. frequency (MHz)]
      Di = 12 [bit rate conversion factor]
TB1 : 00 pa = 4 % [programming voltage current]
      I = 25 mA [maximum current]
      P = 0 V [programming voltage]
TC1 : 00 N = 0 etu [extra guard time]
TD1 : 80 T = T=0 [protocol type]
TD2 : 31 T = T=1 [protocol type]
TA3 : FE IFSC = 254 [information field size]
TB3 : 45 CWT = 43 etu [character waiting time]
      BWT = 15371 etu [block waiting time]
(place for historical bytes)
```

### PIN1, PIN2 and PUK operations

```
>> 00 20 00 01 04 31 32 33 34 00 - VERIFY (PIN1)
<< 90 00 - OK
>> 00 20 00 02 05 31 32 33 34 35 00 - VERIFY (PIN2)
<< 90 00 - OK
>> 00 20 00 00 08 31 32 33 34 35 36 37 38 00 - VERIFY (PUK)
<< 90 00 - OK
// Change PIN1 "1234" => "4321"
>> 00 24 00 01 08 31 32 33 34 34 33 32 31 00 - CHANGE REFERENCE DATA (PIN1)
<< 90 00 - OK
// Change PIN2 "12345" => "54321"
>> 00 24 00 02 0A 31 32 33 34 35 35 34 33 32 31 00 - CHANGE REFERENCE DATA (PIN2)
<< 90 00 - OK
// Change PIN2 "12345678" => "87654321"
>> 00 24 00 00 10 31 32 33 34 35 36 37 38 38 37 36 35 34 33 32 31 00 - CHANGE REFERENCE
DATA (PIN3)
<< 90 00 - OK
// Block PIN1
>> 00 20 00 01 04 31 32 33 34 00 - VERIFY (PIN1)
<< 63 C2 - FAILED (2 tries remaining)
>> 00 20 00 01 04 31 32 33 34 00 - VERIFY (PIN1)
<< 63 C1 - OK (1 tries remaining)
>> 00 20 00 01 04 31 32 33 34 00 - VERIFY (PIN1)
<< 63 C0 - OK (0 tries remaining, blocked)
// Block PIN2
>> 00 20 00 02 05 31 32 33 34 35 00 - VERIFY (PIN2)
```



```
<< 63 C2 - FAILED (2 tries remaining)
>> 00 20 00 02 05 31 32 33 34 35 00 - VERIFY (PIN2)
<< 63 C1 - OK (1 tries remaining)
>> 00 20 00 02 05 31 32 33 34 35 00 - VERIFY (PUK)
<< 63 C0 - OK (0 tries remaining, blocked)
// Unblock PIN1 with pre-verified PUK
>> 00 20 00 00 08 38 37 36 35 34 33 32 31 00 - VERIFY (PUK)
<< 90 00 - OK
>> 00 2C 03 01 00 - RESET RESTY COUNTER
<< 90 00 - OK
// Unblock PIN2 with same command PUK verification and new PIN2 assigning
>> 00 2C 00 02 0C 3 37 36 35 34 33 32 31 31 32 33 34 35 - RESET RESTY COUNTER
<< 90 00 - OK
// Select EF FID 0016 for reading
>> 00 A4 02 0C 02 00 16 00 - SELECT (EF 0016)
<< 90 00 - OK
// Read record 1 containing info for PIN1
>> 00 B2 01 04 00 - READ RECORD
<< 80 01 03 90 01 03 83 02 00 00 90 00 - OK (Tries: max = 3, remaining = 3)
// Read record 2 containing info for PIN2
>> 00 B2 02 04 00 - READ RECORD
<< 80 01 03 90 01 03 83 02 00 00 90 00 - OK (Tries: max = 3, remaining = 3)
// Read record 3 containing info for PUK
>> 00 B2 03 04 00 - READ RECORD
<< 80 01 03 90 01 03 90 00 - OK (Tries: max = 3, remaining = 3)
```

### Navigate to DF FID EEEE<sub>hex</sub>

```
>> 00 A4 00 0C 00 - SELECT (MF)
<< 90 00 - OK
>> 00 A4 01 0C 02 EE EE 00 - SELECT (DF EEEE)
<< 90 00 - OK
```

### Select EF FID 5044<sub>hex</sub> and read all of its contents

```
>> 00 A4 02 0C 02 50 44 00 - SELECT (EF Personal data)
<< 90 00 - OK
>> 00 B2 01 04 00 - READ RECORD (Surname)
<< 4D C4 4E 4E 49 4B 90 00 - OK "MÄNNIK"
>> 00 B2 02 04 00 - READ RECORD (First name 1)
<< 4D 41 52 49 2D 4C 49 49 53 90 00 - OK "MARI-LIIS"
>> 00 B2 03 04 00 - READ RECORD (First name 2)
<< 90 00 - OK ""
>> 00 B2 04 04 00 - READ RECORD (Sex)
<< 4E 90 00 - OK "N"
>> 00 B2 05 04 00 - READ RECORD ()
<< 45 53 54 90 00 - OK "EST"
>> 00 B2 06 04 00 - READ RECORD (Birth date)
<< 30 31 2E 30 31 2E 31 39 37 30 90 00 - OK "01.01.1971"
>> 00 B2 07 04 00 - READ RECORD (Personal identification number)
<< 34 37 31 30 31 30 31 30 30 33 33 90 00 - OK "47101010033" [Seed for CMKs]
>> 00 B2 08 04 00 - READ RECORD (Document number)
<< 41 53 30 30 31 31 31 32 35 90 00 - OK "AS0011125"
>> 00 B2 09 04 00 - READ RECORD (Expiration date)
```



```
<< 30 31 2E 30 32 2E 32 30 31 37 90 00 - OK "01.02.2017"
>> 00 B2 0A 04 00 - READ RECORD (Birth place)
<< 45 45 53 54 49 20 2F 20 45 53 54 90 00 - OK "EESTI / EST"
>> 00 B2 0B 04 00 - READ RECORD (Issuance date)
<< 30 31 2E 30 31 2E 32 30 31 32 90 00 - OK "01.01.2012"
>> 00 B2 0C 04 00 - READ RECORD (Residence permit type)
<< 90 00 - OK ""
>> 00 B2 0D 04 00 - READ RECORD (Notes 1)
<< 90 00 - OK ""
>> 00 B2 0E 04 00 - READ RECORD (Notes 2)
<< 90 00 - OK ""
>> 00 B2 0F 04 00 - READ RECORD (Notes 3)
<< 90 00 - OK ""
>> 00 B2 10 04 00 - READ RECORD (Notes 4)
<< 90 00 - OK ""
```

## Read certificate files

### Read authentication certificate using multiple C-APDUs

```
>> 00 A4 02 0C 02 AA CE - SELECT (EF Authentication certificate)
<< 90 00 - OK
>> 00 B0 00 00 00 - READ BINARY
<< 30 82 05 AD 30 82 03 95 A0 03 02 01 02 02 10 1C 5A 5D 2B EC C2 42 C1 56 E2 C7 0E A9 66
4E 68 30 0D 06 09 2A 86 48 86 F7 0D 01 01 0B 05 00 30 63 31 0B 30 09 06 03 55 04 06 13
02 45 45 31 22 30 20 06 03 55 04 0A 0C 19 41 53 20 53 65 72 74 69 66 69 74 73 65 65 72
69 6D 69 73 6B 65 73 6B 75 73 31 17 30 15 06 03 55 04 61 0C 0E 4E 54 52 45 45 2D 31 30
37 34 37 30 31 33 31 17 30 15 06 03 55 04 03 0C 0E 45 53 54 45 49 44 2D 53 4B 20 32 30
31 35 30 1E 17 0D 31 35 31 32 33 31 32 32 30 30 30 30 5A 17 0D 31 36 31 32 33 30 32 32
30 30 30 30 5A 30 81 9B 31 0B 30 09 06 03 55 04 06 13 02 45 45 31 0F 30 0D 06 03 55 04
0A 0C 06 45 53 54 45 49 44 31 17 30 15 06 03 55 04 0B 0C 0E 61 75 74 68 65 6E 74 69 63
61 74 69 6F 6E 31 26 30 24 06 03 55 04 03 0C 1D 4D C3 84 4E 4E 49 4B 2C 90 00 - OK
```

File length:  $5AD_{hex} + 4 = 5B1_{hex}$

Parts to read:  $(5B1_{hex} + 100_{hex}) / 100_{hex} = 6$

Last part length:  $5B1_{hex} \% 100_{hex} = B1_{hex}$

```
>> 00 B0 01 00 00 - READ BINARY (2nd part)
<< 4D 41 52 49 2D 4C 49 49 53 2C 34 37 31 30 31 30 31 30 30 33 33 31 10 30 0E 06 03 55 04
04 0C 07 4D C3 84 4E 4E 49 4B 31 12 30 10 06 03 55 04 2A 0C 09 4D 41 52 49 2D 4C 49 49
53 31 14 30 12 06 03 55 04 05 13 0B 34 37 31 30 31 30 31 30 30 33 33 30 82 01 22 30 0D
06 09 2A 86 48 86 F7 0D 01 01 01 05 00 03 82 01 0F 00 30 82 01 0A 02 82 01 01 00 94 C6
4F D7 E1 F9 95 B6 A1 CD EE E0 4A A5 C2 3A 53 C3 71 00 7C 8E 65 0C 24 98 C4 5E EA 53 20
2D 59 92 A2 1A 37 41 F6 53 27 F7 6D 3D A2 82 99 7E D6 6C 50 3D EC BE 05 07 4B 14 6D 2D
F6 08 8F A8 87 5B 99 15 10 9D 33 5B DC 84 D1 E2 85 B1 2C BB 89 20 8B A3 6E 11 8A AF 54
00 7E EF A3 E4 A2 67 40 4D 0F 74 5D 0F CE 9C DB D9 EB AB 06 C0 93 31 BE 87 EB C9 F6 32
8B 32 15 CC 3B 08 10 F7 4B A5 0A A0 DF 16 13 9F 04 94 B1 FF 77 7A CD 02 90 00 - OK
>> 00 B0 02 00 00 - READ BINARY (3rd part)
<< 67 7B BE 4F A6 77 91 C8 AD CA 3C 43 D0 4D 76 36 CE F5 AB BB 44 CE BD 4A 1A 8E 03 10 1E
D8 DA D4 D6 2B 28 42 21 30 8E 54 DA CF 74 73 4E 53 6D A8 BB 48 82 63 8B 6A 4A 73 DD 20
3D C5 3C CF 44 A8 DB 88 F2 56 23 7D 4F 2C 60 A3 BE 10 62 EE 37 1F D0 61 B7 D4 EA 9D C3
C7 02 51 FA 7B DB 4D 94 41 33 11 07 F7 DB 4D 29 A2 B0 44 3C B2 73 75 02 00 B6 8B 02 03
01 00 01 A3 82 01 22 30 82 01 1E 30 09 06 03 55 1D 13 04 02 30 00 30 0E 06 03 55 1D 0F
01 01 FF 04 04 03 02 04 B0 30 3B 06 03 55 1D 20 04 34 30 32 30 30 06 09 2B 06 01 04 01
CE 1F 01 01 30 23 30 21 06 08 2B 06 01 05 05 07 02 01 16 15 68 74 74 70 73 3A 2F 2F 77
77 77 2E 73 6B 2E 65 65 2F 63 70 73 30 24 06 03 55 1D 11 04 1D 30 1B 81 19 6D 61 72 69
2D 6C 69 69 73 2E 6D 61 6E 6E 69 6B 40 65 65 73 74 69 2E 65 65 30 1D 06 90 00 - OK
>> 00 B0 03 00 00 - READ BINARY (4th part)
<< 03 55 1D 0E 04 16 04 14 BF A5 73 79 75 8B 10 75 67 23 DE 00 D0 2F 31 CE CD 1D 73 19 30
20 06 03 55 1D 25 01 01 FF 04 16 30 14 06 08 2B 06 01 05 05 07 03 02 06 08 2B 06 01 05
05 07 03 04 30 1F 06 03 55 1D 23 04 18 30 16 80 14 B3 AB 88 BC 99 D5 62 A4 85 2A 08 CD
B4 1D 72 3B 83 72 47 51 30 3C 06 03 55 1D 1F 04 35 30 33 30 31 A0 2F A0 2D 86 2B 68 74
74 70 3A 2F 2F 77 77 77 2E 73 6B 2E 65 65 2F 63 72 6C 73 2F 65 73 74 65 69 64 2F 65 73
```



```
74 65 69 64 32 30 31 35 2E 63 72 6C 30 0D 06 09 2A 86 48 86 F7 0D 01 01 0B 05 00 03 82
02 01 00 AE 26 B3 98 E4 3B FF 40 03 22 11 0A 72 E6 70 1F D3 82 5E 12 6F 76 3F 7C 43 57
57 53 68 10 20 76 7B 3A 1D FE 67 A1 29 5B F8 9E 33 61 95 FD C2 E0 B3 E9 1C 92 18 B5 CA
45 C2 71 93 81 48 24 F2 76 9C 3F 83 05 99 59 7A A1 52 B8 65 ED CD 82 81 90 00 - OK
>> 00 B0 04 00 00 - READ BINARY (5th part)
<< D4 A8 36 54 79 AF 87 53 05 11 47 1A BE FC 67 CF 84 F0 47 80 27 3C 66 60 8F AB 01 51 C2
7E 73 FD 0E 0B 4F 33 42 28 09 08 96 59 38 E3 C2 09 3B FD 6B 63 D2 9E D9 C8 0A 4F 04 81
8C 24 12 1D 62 D7 0C DB 03 51 20 F7 90 59 26 EA AD EC B7 A0 76 8E 67 64 FE 57 98 A9 02
91 06 32 90 67 CB 1E 16 5A 47 D5 61 17 E3 B9 A0 AF 8B 79 01 48 04 18 F0 06 AF 86 F1 51
D9 65 11 C0 CF 73 3D 5F A7 FC E2 81 A4 A5 51 46 5D 1B 28 D1 20 1F 15 00 E9 19 07 35 23
3D A2 39 79 D5 BD A6 CE D6 22 BF 17 D1 ED 5C 3A 46 DA B0 AD 0E 44 BF D5 AE DF 1D 8C 98
38 75 EA 99 23 7E AB D3 31 D6 64 5A 99 4D EC 74 AE A3 64 99 D9 6D CB 9F 15 F4 F1 53 7E
51 F6 EF 3D 00 EB CA 4D 63 5E 91 D8 CA 91 EB 30 77 AE E2 1F 60 CC 38 78 08 A1 9E 16 6C
0A 09 98 2A E1 C4 64 B2 8A 96 0B 86 07 5C 42 E8 08 B6 7C 5F 9C A6 A3 18 90 00 - OK
>> 00 B0 05 00 B1 - READ BINARY (6th part)
<< DB 55 E2 3D 4D EB 78 18 0C 2B 72 8F 7E 91 A2 1E D4 E9 EE FA D3 7A 3A 3E C1 46 33 E0 21
DA 6C 3C D8 FE 64 1B F7 F8 B3 72 58 B0 03 B2 32 81 78 8D C1 E5 C1 2F BE D7 8F F1 CD 06
31 43 24 19 66 9C 86 AF 90 BE F7 E6 12 B4 55 93 14 C7 C5 FA F3 1F E3 48 FD 93 6E FC 57
AB 2B A1 08 F3 A2 1F 7D 79 ED 5E AC 2B 4A 13 9E 8F 3D 74 F3 59 B8 95 3B 96 86 30 89 C6
76 AF F8 3D 76 80 F7 78 72 A9 B7 D5 54 A8 6B 72 30 4C C1 9A 2C C0 A4 0D 4E 84 B0 9E 3C
EA CB EA B2 6E 2A 45 F6 CC 3E 99 38 44 3B 3D 5F 12 49 07 76 12 F2 94 B1 9C DA 3C 8E D2
BB E2 86 90 00 - OK
## Derive active authentication key from certificate ##
RSA public modulus:
F82F7ECD9FB168FE8550FE10E383C25D9AD460EC4BC64B9994D053A3EBD56BD651B75090A80883CAE5E1F6A
62E9E395B7F0E6DF444227241C84CAE8AC0AE5E728E6C7CC9634FD0930340F94BCC8B04E0D9ADD14423B1F0
F8217566B64C6645FBB7E4F31671FB6DB345262976524F4B564A074F906617E77BF00897DCE78FC00F0E84B
2F7C4988D0CB15D1A9E8ABAF66C383FF0A68A7956C277CE0210436F142FC60BF1CEDE88B3C607B41B544E4D
67171333BEEF618666B04D9A02A24FE8E0D75A9A9C95674D9E66416F5B400FD167AE71A0D48057E8BA401EE
68E9A63595178E4978594427C19068B90192FA23EBE6C36A53AA7078CEC1925CA87CD4FDAF197
RSA public exponent: 40000081
```

## Read signature certificate using extended C-APDU

```
>> 00 A4 02 0C 02 DD CE - SELECT (EF Signature certificate)
<< 90 00 - OK
>> 00 B0 00 00 00 08 00 - READ BINARY (EXTENDED)
<< 30 82 05 AD 30 82 03 95 A0 03 02 01 02 02 10 1C 5A 5D 2B EC C2 42 C1 56 E2 C7 0E A9 66
4E 68 30 0D 06 09 2A 86 48 86 F7 0D 01 01 0B 05 00 30 63 31 0B 30 09 06 03 55 04 06 13
02 45 45 31 22 30 20 06 03 55 04 0A 0C 19 41 53 20 53 65 72 74 69 66 69 74 73 65 65 72
69 6D 69 73 6B 65 73 6B 75 73 31 17 30 15 06 03 55 04 61 0C 0E 4E 54 52 45 45 2D 31 30
37 34 37 30 31 33 31 17 30 15 06 03 55 04 03 0C 0E 45 53 54 45 49 4D 53 4B 20 32 30
31 35 30 1E 17 0D 31 35 31 32 33 31 32 32 30 30 30 5A 17 0D 31 36 31 32 33 30 32 32
30 30 30 30 5A 30 81 9B 31 0B 30 09 06 03 55 04 06 13 02 45 45 31 0F 30 0D 06 03 55 04
0A 0C 06 45 53 54 45 49 44 31 17 30 15 06 03 55 04 0B 0C 0E 61 75 74 68 65 6E 74 69 63
61 74 69 6F 6E 31 26 30 24 06 03 55 04 03 0C 1D 4D C3 84 4E 4E 49 4B 2C 4D 41 52 49 2D
4C 49 49 53 2C 34 37 31 30 31 30 31 30 30 33 33 31 10 30 0E 06 03 55 04 04 0C 07 4D C3
84 4E 4E 49 4B 31 12 30 10 06 03 55 04 2A 0C 09 4D 41 52 49 2D 4C 49 49 53 31 14 30 12
06 03 55 04 05 13 0B 34 37 31 30 31 30 31 30 30 33 33 30 82 01 22 30 0D 06 09 2A 86 48
86 F7 0D 01 01 05 00 03 82 01 0F 00 30 82 01 0A 02 82 01 01 00 9F 6C 78 11 D6 62 F9
7B 7C 8C C3 8F 41 CB 71 60 E9 25 B4 8B EB 6C DF DD 03 AE 3B 4A 6E CA 16 4D 23 DD 3C 4F
46 FD 72 06 66 69 AC BC E6 31 B0 D5 F1 2B D9 BC BF 4F 3F F7 1F B9 77 54 49 9D 8E A9 8B
9B 6B E6 30 16 C7 8F 60 1E 70 65 4A 97 FE 63 87 10 80 AE 8E C2 7C A0 11 C6 97 23 2E F8
FA BA 8E F8 89 3E AF 08 8D A1 1C EE F4 B7 3E B1 09 05 37 3B 64 A8 14 2D 00 BF 45 31 41
10 93 C6 15 14 1F 60 53 38 33 0E 99 9A 5E 52 F9 7D 80 42 E8 50 55 F1 F0 47 3A AD B2 C9
2F A6 E1 D5 98 58 6E 77 B8 51 7C 9B 49 8C AA FE EE 49 F4 F6 4E B5 D6 CF C1 C7 63 78 77
8A 70 E7 63 E0 A4 02 F3 86 BC D0 CD B5 E9 95 E9 DF C3 10 B6 FD B7 67 C4 77 8B E8 5C 10
BD E3 F2 A4 FD 6B AC 1A F4 67 08 4B 5D 91 42 D9 90 A2 6D 79 CC 28 41 7C 15 05 AE 56
E7 88 B8 D0 2B A9 08 F5 15 D7 69 4C 7D 2F 0B 9B AB 02 03 01 00 01 A3 82 01 22 30 82 01
1E 30 09 06 03 55 1D 13 04 02 30 00 30 0E 06 03 55 1D 0F 01 01 FF 04 04 03 02 04 B0 30
3B 06 03 55 1D 20 04 34 30 32 30 30 06 09 2B 06 01 04 01 CE 1F 01 01 30 23 30 21 06 08
2B 06 01 05 05 07 02 01 16 15 68 74 74 70 73 3A 2F 77 77 72 7E 73 6B 2E 65 65 2F 63
70 73 30 24 06 03 55 1D 11 04 1D 30 1B 81 19 6D 61 72 69 2D 6C 69 69 73 2E 6D 61 6E 6E
69 6B 40 65 65 73 74 69 2E 65 65 30 1D 06 03 55 1D 0E 04 16 04 14 BF A5 73 79 75 8B 10
75 67 23 DE 00 D0 2F 31 CE CD 1D 73 19 30 20 06 03 55 1D 25 01 01 FF 04 16 30 14 06 08
2B 06 01 05 05 07 03 02 06 08 2B 06 01 05 05 07 03 04 30 1F 06 03 55 1D 23 04 18 30 16
80 14 B3 AB 88 BC 99 D5 62 A4 85 2A 08 CD B4 1D 72 3B 83 72 47 51 30 3C 06 03 55 1D 1F
04 35 30 33 30 31 A0 2F A0 2D 86 2B 68 74 74 70 3A 2F 2F 77 77 77 2E 73 6B 2E 65 65 2F
```







## Card application general operations

### Calculate response for TLS challenge

```
>> 00 22 F3 01 00 - MANAGE SECURITY ENVIRONMENT (Select active keys)
<< 90 00 - OK
>> 00 20 00 01 04 34 33 32 31 00 - VERIFY (PIN1)
<< 90 00 - OK
Random 24 bytes: E2 CC 6D 68 16 0C 07 AC C1 EB 33 1A 77 96 3B 70 54 58 1F 5A E1 E6 B2 29
>> 00 88 00 00 18 E2 CC 6D 68 16 0C 07 AC C1 EB 33 1A 77 96 3B 70 54 58 1F 5A E1 E6 B2 29
00 - INTERNAL AUTHENTICATE
<< 3C 45 6F 75 7E CD 9F 9D 37 A9 2E A7 3E F8 13 C9 F0 EC 58 16 A3 59 C2 9F 9A 81 5C FD 87
38 1D A9 F3 41 D8 A8 F4 30 C5 00 47 EF A4 84 AD F7 F7 B1 CB E7 3A BD A9 12 93 24 19 F8
89 D9 52 2F 78 E9 8A 0E 62 EF FA C3 BB 6E F9 0B 02 20 71 2B B1 AF DF DB 10 02 AD 58 FF
54 CD E3 F0 E5 4B 22 3F 5F 1E D9 C1 67 62 C2 53 D8 73 25 AA 80 C3 7B 7F 51 20 E4 FF 0C
02 7A 07 EE D2 D1 47 28 B2 D3 77 36 DD 4F 1C 35 E3 D0 5A 4F FC 02 76 55 54 C5 6F 38 0E
EC F1 21 F1 38 E7 B6 1B 26 FE 9B 2E 11 7D EA 38 0F 3A DD B6 39 36 E4 87 37 22 F0 9C 17
87 8F C0 2E 04 C8 DD BC A3 2C 51 DD 2B AA 83 41 6D F5 68 7D AC 57 E9 FF 71 65 D0 2B B6
CF A7 3A 54 F1 DE D7 7E 90 FC 9A 18 06 35 30 AE 64 2C DC B5 FB 6D 3D B5 07 27 C6 9D C4
6C 31 AC B8 10 EB AC 17 5A 1B FB 96 12 AF 85 77 5E 1B 7A FF DE 90 02 2E 90 00 - OK
Decrypted_Random = RSAPub.decrypt(R-APDU data): E2 CC 6D 68 16 0C 07 AC C1 EB 33 1A 77 96
3B 70 54 58 1F 5A E1 E6 B2 29
Random == Decrypted_Random (true)
```

### Calculate electronic signature from pre-calculated SHA1 hash

```
>> 00 22 F3 01 00 - MANAGE SECURITY ENVIRONMENT (Select active keys)
<< 90 00 - OK
>> 00 20 00 02 05 31 32 33 34 35 00 - VERIFY (PIN2)
<< 90 00 - OK
SHA1("MARI-LIIS MÄNNIK"): F8 E5 40 13 C8 61 C2 A7 46 3E 50 B9 BD 4C E3 2D 64 9D EF 9C
>> 00 2A 9E 9A 23 30 21 30 09 06 05 2B 0E 03 02 1A 05 00 04 14 F8 E5 40 13 C8 61 C2 A7 46
3E 50 B9 BD 4C E3 2D 64 9D EF 9C 00 - PERFORM SECURITY OPERATION (COMPUTE DIGITAL
SIGNATURE)
<< 7D 7F D2 85 EC AE CD E8 C7 05 0E 5E F8 C5 EF 86 9A D3 27 E9 7F ED 58 C8 93 5A 2D F0 B1
7F C8 39 08 6C 19 9E 19 0C 77 F7 2F 9E DE 85 1B FA 76 D9 A1 9F 3E 89 5B 03 EF FF F1 42
A9 7B 66 5E A4 95 7E 9E CB 58 8E E9 F5 0F B1 61 8D 03 86 C8 59 8A BD 77 18 5F 30 6A 8B
F3 3A FE 4D 80 AC 64 CC 5D B9 21 0B EF BD 96 D5 6E ED 17 E4 F6 51 C5 E0 B0 67 BB 0E 10
69 14 9A 90 DF 57 6B 31 7A AB F2 DB FE 7F 83 B7 62 6C 06 05 AE 23 BA 0C F2 61 87 DB 98
D9 67 90 93 93 0B 26 1A A3 BF 4A F1 BD 4F A7 42 99 87 2A 9D 78 9A 45 C9 D6 F3 CC 71 EE
9E 2F EC D7 2B 9C 89 8D 3F 17 CA 0E 99 54 5A 31 FE D0 CB 64 69 16 71 63 8B 07 09 F6 4E
DA D9 E7 BD 19 A0 6D BC 6E 19 80 1E 8B A2 FE 29 63 02 73 83 30 79 29 24 F4 C5 E6 75 42
6F D1 82 0F 9E DB 30 AE 9D 5F 3F 05 19 58 75 BD 24 21 40 56 90 1A AB D4 90 00 - OK
```

### Perform deciphering operation

```
>> 00 22 41 A4 05 83 03 80 11 00 00 - MANAGE SECURITY ENVIRONMENT (Select active keys)
<< 90 00 - OK
>> 00 20 00 01 04 34 33 32 31 00 - VERIFY (PIN1)
<< 90 00 - OK
Input_data = ("MARI-LIIS MÄNNIK")
RSAPub.encrypt(Input_data) for command DECIPHER
>> 00 2A 80 86 00 01 01 00 0E CD 8C 82 FB 7F CB 49 A8 21 47 7C 25 2E E7 8D FE 90 AF F7 8B
28 18 AE 54 5E C2 A1 F7 1D CA 43 AF FD 7D 99 60 87 94 B4 01 03 CC 62 1C 55 D4 82 1F 68
6B 2A 64 3B 5F 1E 0E FE 8E C6 98 97 C5 F7 9E F2 A4 E3 ED FC 60 A1 52 52 06 17 2A E0 AF
F9 64 2A 89 34 69 0E 72 04 F0 97 31 D9 F9 71 FA 13 3A C4 56 E0 05 9A 9D BF D9 40 8D 2B
15 85 C0 68 31 C2 30 D8 C4 AC 43 1B AC 88 6D 32 D3 88 71 99 12 CF 6B 32 96 99 83 7D 1E
C4 F8 1C 7B 42 7C 9E 31 FB 60 01 F2 D7 8E 32 39 36 8F A7 55 17 90 F2 F1 0C E4 49 53 D0
7A 81 BA 5A 96 E7 7F 33 C1 F9 EB F7 91 ED 53 5D 48 CA 3C 06 D5 4F E9 E3 35 5B 1D E7 C1
71 15 A7 2B 60 C4 15 E6 3E 6B 34 A8 94 C5 A8 22 9C FA 55 0F 91 3C D6 05 F8 CD 7A 82 A4
F7 3F A8 63 97 45 B2 35 02 EA 73 46 CE 8A CF 77 8A BF 72 6C 4E A8 49 F1 D1 13 68 F2 80
80 08 C3 00 00
<< 4D 41 52 49 2D 4C 49 49 53 20 4D C4 4E 4E 49 4B 90 00 - OK
```



Input\_data == R-APDU data (true)

## Card application managing operations

### Replace cardholder PINs/PUK codes

```
// Use master CMK_PIN
// Calculate cardholder CMK. "47101010033" as seed, from EF 5044 record 7.
SHA1("47101010033") = 74 B5 97 30 6B C5 9B 67 2F B1 64 B2 0F 36 23 3A B3 35 37 0C
Take 16 leftmost bytes of calculated SHA1: 74 B5 97 30 6B C5 9B 67 2F B1 64 B2 0F 36
                                           23 3A
CMK_PIN.encrypt(74 B5 97 30 6B C5 9B 67 2F B1 64 B2 0F 36 23 3A) =
                                                                    A6 5E 60 AE 5A E4 74 F0
                                                                    BC BC 0A AA 3A AE 9E DC

Set off LSB bits of every byte: A65E60AE5AE474F0BCBC0AAA3AAE9EDC (cardholder CMK_PIN)
>> 00 84 00 00 08 - GET CHALLENGE
<< 26 03 6B B7 F6 A8 ED C2 90 00 - OK
RND_ICC: 26036BB7F6A8EDC2
RND_IFD: E957BA5E87DD753C
K_IFD: A3 A3 9B 2B 97 33 A9 FE 50 99 7C 53 EA F8 C2 61 4C 1B 95 BB F5 E1 A1 99 57 95 D2 18
      F9 CF 5D 27
CMK.encrypt(RND_IFD || RND_ICC || K_IFD) = 8A A0 1B 13 37 84 78 A9 05 4C 7C 79 50 12 79 4C
                                           74 03 11 53 2B 85 01 5A 6B 26 8F F0 64 A3 EA B7
                                           E5 B9 FD 73 49 02 1A 7B F4 19 74 0C A4 A5 40 51
>> 00 82 00 01 30 8A A0 1B 13 37 84 78 A9 05 4C 7C 79 50 12 79 4C 74 03 11 53 2B 85 01 5A
      6B 26 8F F0 64 A3 EA B7 E5 B9 FD 73 49 02 1A 7B F4 19 74 0C A4 A5 40 51 30
<< D6 80 C0 23 1D 74 F4 0C 6F B9 60 66 F2 5C E4 1D C6 B6 E7 53 22 9A E9 15 30 B7 5D 45 09
      EC 03 1B 12 0A 01 A9 52 48 56 CE 8C 44 8A AA 5C 0D 26 43 90 00
CMK.decrypt(chip_response) = 26 03 6B B7 F6 A8 ED C2 E9 57 BA 5E 87 DD 75 3C D0 E2 4A 7C
                             05 21 C7 07 A1 6F FC 58 A7 64 41 51 9E 16 0E C4 B6 35 EC C7
                             F9 CE A8 93 03 18 DF BB
_RND_ICC: 26036BB7F6A8EDC2
_RND_IFD: E957BA5E87DD753C
RND_IFD == _RND_IFD (true)
K_ICC: D0 E2 4A 7C 05 21 C7 07 A1 6F FC 58 A7 64 41 51 9E 16 0E C4 B6 35 EC C7 F9 CE A8 93
      03 18 DF BB
SK = K_IFD XOR _K_ICC : 73 41 D1 57 92 12 6E F9 F1 F6 80 0B 4D 9C 83 30 D2 0D 9B 7F 43 D4
      4D 5E AE 5B 7A 8B FA D7 82 9C
SSC = RND_IFD[4..7] || RND_ICC[4..7]: 87 DD 75 3C F6 A8 ED C2
SK1 = SK[0..15] : 73 41 D1 57 92 12 6E F9 F1 F6 80 0B 4D 9C 83 30
SK2 = SK[16..31] : D2 0D 9B 7F 43 D4 4D 5E AE 5B 7A 8B FA D7 82 9C
## Mutual authentication successful ##

// Secure the command
CLA = CLA | 0C: 0C
SSC(87 DD 75 3C F6 A8 ED C2) + 1 = 87 DD 75 3C F6 A8 ED C3
Data = Data || ISO9797 method 2 padding: 31 32 33 34 31 32 33 34 35 31 32 33 34 35 36 37
                                           38 80 00 00 00 00 00 00
Cryptogram = SK1.encrypt(data, IV(SSC)) = 35 CC 72 19 34 AD E6 A4 E5 33 28 9E B7 50 F4 83
                                           05 6A 2E 1D A1 F8 10 93

// Prepare MACData
Append header: 0C 05 00 00
Append 80000000: 0C 05 00 00 80 00 00 00
Wrap Cryptogram into TLV with tag 87.
Data = Tag(87) || Length || Value(Cryptogram): 87 19 01 35 CC 72 19 34 AD E6 A4 E5 33
                                                28 9E B7 50 F4 83 05 6A 2E 1D A1 F8 10
                                                93
```



```
MACData = Append ISO9797 method 2 padding: 0C 05 00 00 80 00 00 00 87 19 01 35 CC 72
                                              19 34 AD E6 A4 E5 33 28 9E B7 50 F4 83 05
                                              6A 2E 1D A1 F8 10 93 80 00 00 00 00

// Calculate MAC
SK2Key1 = SK2[0..7]: D2 0D 9B 7F 43 D4 4D 5E
SK2Key1 = SK2[8..15]: AE 5B 7A 8B FA D7 82 9C
MAC = SK2Key1.CBC_encrypt(MAC Data, IV(SSC)): 01 71 5C E8 C9 DB 03 60
MAC = SK2Key2.decrypt(MAC): E3 49 9B 2C 55 EA 8E 1B
MAC = SK2Key1.encrypt(MAC): 7C 56 5F 78 4D 32 E9 A6
// Wrap MAC into TLV with tag 8E.
MAC = Tag(8E) || Length || Value(MAC): 8E 08 7C 56 5F 78 4D 32 E9 A6
// Append MAC to Data
Data = Data || MAC: 87 19 01 35 CC 72 19 34 AD E6 A4 E5 33 28 9E B7 50 F4 83 05 6A 2E 1D
                  A1 F8 10 93 8E 08 7C 56 5F 78 4D 32 E9 A6
>> 0C 05 00 00 25 87 19 01 35 CC 72 19 34 AD E6 A4 E5 33 28 9E B7 50 F4 83 05 6A 2E 1D A1
    F8 10 93 8E 08 7C 56 5F 78 4D 32 E9 A6 00 - SECURE REPLACE PINS
<< 99 02 90 00 8E 08 DA 14 A5 89 A4 68 B9 44 90 00 - OK
// Verify MAC and decrypt
SSC(87 DD 75 3C F6 A8 ED C3) + 1 = 87 DD 75 3C F6 A8 ED C4
MAC: DA 14 A5 89 A4 68 B9 44
// Prepare MACData
Append R-APDU data without MAC TLV(8E): 99 02 90 00
MACData = Append ISO9797 method 2 padding: 99 02 90 00 80 00 00 00
_MAC = SK2Key1.CBC_encrypt(Data, IV(SSC)): 4F 35 23 C1 BE 09 A5 9D
_MAC = SK2Key2.decrypt(_MAC): D5 B8 6E C6 24 CD 23 72
_MAC = SK2Key1.encrypt(_MAC): DA 14 A5 89 A4 68 B9 44
MAC == _MAC (true)
```

## Generate new key pair

```
>> 00 20 00 01 04 31 32 33 34 00 - VERIFY (PIN1)
<< 90 00 - OK

// Use master CMK_KEY
// Calculate cardholder CMK. "47101010033" as seed, from EF 5044 record 7.
SHA1("47101010033") = 74 B5 97 30 6B C5 9B 67 2F B1 64 B2 0F 36 23 3A B3 35 37 0C
Take 16 leftmost bytes of calculated SHA1: 74 B5 97 30 6B C5 9B 67 2F B1 64 B2 0F 36 23
                                           3A
CMK_KEY.encrypt(74 B5 97 30 6B C5 9B 67 2F B1 64 B2 0F 36 23 3A) =
                                                                82 9C AC 1E DE DA 26 90
                                                                BA 88 58 76 58 48 BA DC

Set off LSB bits of every byte: 829CAC1EDED A2690BA8858765848BADC (cardholder CMK_KEY)
>> 00 84 00 00 08 - GET CHALLENGE
<< C6 48 63 14 A2 DF 2D 22 90 00 - OK
RND_ICC: C6 48 63 14 A2 DF 2D 22
Generate RND_IFD: E6 D1 E5 54 05 18 09 B6
Generate K_IFD: 5E 70 34 88 41 C1 F8 CE 8C 71 E1 AD 81 20 C4 99 A8 B2 10 75 0F A3 F9 1F
                13 DB FE FE 34 9E BB 39
CMK.encrypt(RND_IFD || RND_ICC || K_IFD) = 16 FC DD 22 26 EA 77 7B 13 C0 AF F3 84 FF B2 B8
                                           3F 47 E7 D8 CF FE 14 6A 2E 48 7D 53 BE 2F 18 3A
                                           C0 7B 77 74 EB 2F 57 77 1E 47 BC BC 98 6B DC 26
>> 00 82 00 02 30 16 FC DD 22 26 EA 77 7B 13 C0 AF F3 84 FF B2 B8 3F 47 E7 D8 CF FE 14 6A
    2E 48 7D 53 BE 2F 18 3A C0 7B 77 74 EB 2F 57 77 1E 47 BC BC 98 6B DC 26 30 - MUTUAL
    AUTHENTICATE
<< EA DF 90 DD DD FA 46 98 25 0B 9B BC A6 E9 ED 10 BB 12 FE 9F 44 BB 0E 52 7E C9 36 1A F5
    06 45 08 43 4E 42 D0 AA 3A 7E 4A 32 A6 8D 8C F4 91 52 F8 90 00 - OK
```



```
CMK.decrypt(chip_response) = C6 48 63 14 A2 DF 2D 22 E6 D1 E5 54 05 18 09 B6 FB A7 1D 39
                                08 AD 1A EF BC F4 3F 16 0F 5A C7 11 8F B4 A4 BA E0 AB 27 07
                                99 64 8F FF 71 A1 5F 75

_RND_ICC: C6 48 63 14 A2 DF 2D 22
_RND_IFD: E6 D1 E5 54 05 18 09 B6
RND_IFD == _RND_IFD (true)
K_ICC:   FB A7 1D 39 08 AD 1A EF BC F4 3F 16 0F 5A C7 11 8F B4 A4 BA E0 AB 27 07 99 64 8F
        FF 71 A1 5F 75

SK = K_IFD XOR _K_ICC: A5 D7 29 B1 49 6C E2 21 30 85 DE BB 8E 7A 03 88 27 06 B4 CF EF 08
                      DE 18 8A BF 71 01 45 3F E4 4C

SSC = RND_IFD[4..7] || RND_ICC[4..7]: 05 18 09 B6 A2 DF 2D 22
SK1 = SK[0..15]: A5 D7 29 B1 49 6C E2 21 30 85 DE BB 8E 7A 03 88
SK2 = SK[16..31]: 27 06 B4 CF EF 08 DE 18 8A BF 71 01 45 3F E4 4C

## Mutual Authentication successful ##

// Secure the command
CLA = CLA | 0C: 0C
SSC(05 18 09 B6 A2 DF 2D 22) + 1 = 05 18 09 B6 A2 DF 2D 23

// Prepare MACData
    Append header: 0C 06 01 00
    Append 80000000: 0C 06 01 00 80 00 00 00
    MACData = Append ISO9797 method 2 padding: 0C 06 01 00 80 00 00 00 80 00 00 00 00 00
                                                00 00

// Calculate MAC
SK2Key1 = SK2[0..7]: 27 06 B4 CF EF 08 DE 18
SK2Key2 = SK2[8..15]: 8A BF 71 01 45 3F E4 4C
MAC = SK2Key1.CBC_encrypt(MACData, IV(SSC)): 30 08 99 FC FC FB A4 6E
MAC = SK2Key2.decrypt(MAC): 3F 33 44 B9 A0 70 C2 CC
MAC = SK2Key1.encrypt(MAC): D3 A6 AA F2 4C 4A 68 F4
// Wrap MAC into TLV with tag 8E.
    MAC = Tag(8E) || Length || Value(MAC): 8E 08 D3 A6 AA F2 4C 4A 68 F4
Data = Data || MAC: 8E 08 D3 A6 AA F2 4C 4A 68 F4
>> 0C 06 01 00 00 00 0A 8E 08 D3 A6 AA F2 4C 4A 68 F4 00 00 - SECURE GENERATE KEY
<< 87 82 01 10 96 AE D2 2E 44 A0 EB 46 E5 3C 11 DE E1 5F 76 A1 E0 37 19 E0 BD 60 FD C3 44
    F2 E8 6F 44 F3 66 04 55 FE 7D D2 8B F1 CB 8C D8 04 92 3D CC 0D BC FB 16 FA 41 7E 8F 1B
    E4 8F 6B 74 21 2B 05 B8 BC FE CB 40 91 84 2C C6 7D 1A 8A 26 59 B1 D8 41 F1 1F 3A 40 65
    E7 AD C5 7F BD 67 0D 13 B4 AA 44 D8 5D 78 8F 86 28 0F 5A 5E A7 E1 0E A3 79 5D C5 B5 CB
    78 C4 93 DC 7D E5 E8 6E 72 2F 4C 23 89 0C F8 CA 0B 8C 08 59 36 E3 F2 07 4B E4 CD B9 3E
    26 93 12 D3 34 01 1A 2A 6B 63 B8 7A 1E A7 A1 CE DD E5 5F 41 49 8D 1B 74 5A 4E 9D EF E7
    CB 39 31 3E 8B 2E EC F9 37 FB 3B 9B 47 AF DE 86 94 4E 4D FC 30 95 A0 8B B1 3D D1 AF B6
    FC 9D DD 2D B3 6F 18 FB B5 60 25 51 45 C9 5E 86 6D 94 FF DB BE 29 9A 1E C2 CC AF F3 51
    1C 7E 06 D3 BF 5E FD E1 B2 62 34 62 4D 44 BA A9 25 7A B4 04 2B 7F B9 55 4F 8E 54 DC 32
    8D A7 87 88 24 6F 59 C5 30 51 0F 5C 5F 8A 6F 8E 08 58 B6 69 A2 DA 1B 62 15 90 00 - OK

// Verify MAC and decrypt
SSC(05 18 09 B6 A2 DF 2D 23) + 1 = 05 18 09 B6 A2 DF 2D 24
MAC: 58 B6 69 A2 DA 1B 62 15

// Prepare MACData
    Append R-APDU data without MAC TLV(8E):
        87 82 01 10 96 AE D2 2E 44 A0 EB 46 E5 3C 11 DE E1 5F 76 A1 E0 37 19 E0 BD
        60 FD C3 44 F2 E8 6F 44 F3 66 04 55 FE 7D D2 8B F1 CB 8C D8 04 92 3D CC 0D
        BC FB 16 FA 41 7E 8F 1B E4 8F 6B 74 21 2B 05 B8 BC FE CB 40 91 84 2C C6 7D
        1A 8A 26 59 B1 D8 41 F1 1F 3A 40 65 E7 AD C5 7F BD 67 0D 13 B4 AA 44 D8 5D
        78 8F 86 28 0F 5A 5E A7 E1 0E A3 79 5D C5 B5 CB 78 C4 93 DC 7D E5 E8 6E 72
        2F 4C 23 89 0C F8 CA 0B 8C 08 59 36 E3 F2 07 4B E4 CD B9 3E 26 93 12 D3 34
        01 1A 2A 6B 63 B8 7A 1E A7 A1 CE DD E5 5F 41 49 8D 1B 74 5A 4E 9D EF E7 CB
        39 31 3E 8B 2E EC F9 37 FB 3B 9B 47 AF DE 86 94 4E 4D FC 30 95 A0 8B B1 3D
        D1 AF B6 FC 9D DD 2D B3 6F 18 FB B5 60 25 51 45 C9 5E 86 6D 94 FF DB BE 29
        9A 1E C2 CC AF F3 51 1C 7E 06 D3 BF 5E FD E1 B2 62 34 62 4D 44 BA A9 25 7A
        B4 04 2B 7F B9 55 4F 8E 54 DC 32 8D A7 87 88 24 6F 59 C5 30 51 0F 5C 5F 8A
        6F
```



```
MACData = Append ISO9797 method 2 padding:
87 82 01 10 96 AE D2 2E 44 A0 EB 46 E5 3C 11 DE E1 5F 76 A1 E0 37 19 E0 BD
60 FD C3 44 F2 E8 6F 44 F3 66 04 55 FE 7D D2 8B F1 CB 8C D8 04 92 3D CC 0D
BC FB 16 FA 41 7E 8F 1B E4 8F 6B 74 21 2B 05 B8 BC FE CB 40 91 84 2C C6 7D
1A 8A 26 59 B1 D8 41 F1 1F 3A 40 65 E7 AD C5 7F BD 67 0D 13 B4 AA 44 D8 5D
78 8F 86 28 0F 5A 5E A7 E1 0E A3 79 5D C5 B5 CB 78 C4 93 DC 7D E5 E8 6E 72
2F 4C 23 89 0C F8 CA 0B 8C 08 59 36 E3 F2 07 4B E4 CD B9 3E 26 93 12 D3 34
01 1A 2A 6B 63 B8 7A 1E A7 A1 CE DD E5 5F 41 49 8D 1B 74 5A 4E 9D EF E7 CB
39 31 3E 8B 2E EC F9 37 FB 3B 9B 47 AF DE 86 94 4E 4D FC 30 95 A0 8B B1 3D
D1 AF B6 FC 9D DD 2D B3 6F 18 FB B5 60 25 51 45 C9 5E 86 6D 94 FF DB BE 29
9A 1E C2 CC AF F3 51 1C 7E 06 D3 BF 5E FD E1 B2 62 34 62 4D 44 BA A9 25 7A
B4 04 2B 7F B9 55 4F 8E 54 DC 32 8D A7 87 88 24 6F 59 C5 30 51 0F 5C 5F 8A
6F 80 00 00 00

_MAC = SK2Key1.CBC_encrypt(MACData, IV(SSC)): C5 C3 7A 1F 98 C7 8C A7
_MAC = SK2Key2.decrypt(_MAC): EF DB 3B 5C 66 F0 C8 50
_MAC = SK2Key1.encrypt(_MAC): 58 B6 69 A2 DA 1B 62 15
MAC == _MAC (true)

Unwrap cryptogram from TLV: 96 AE D2 2E 44 A0 EB 46 E5 3C 11 DE E1 5F 76 A1 E0 37 19 E0 BD
60 FD C3 44 F2 E8 6F 44 F3 66 04 55 FE 7D D2 8B F1 CB 8C D8 04 92 3D CC 0D BC FB 16 FA
41 7E 8F 1B E4 8F 6B 74 21 2B 05 B8 BC FE CB 40 91 84 2C C6 7D 1A 8A 26 59 B1 D8 41 F1
1F 3A 40 65 E7 AD C5 7F BD 67 0D 13 B4 AA 44 D8 5D 78 8F 86 28 0F 5A 5E A7 E1 0E A3 79
5D C5 B5 CB 78 C4 93 DC 7D E5 E8 6E 72 2F 4C 23 89 0C F8 CA 0B 8C 08 59 36 E3 F2 07 4B
E4 CD B9 3E 26 93 12 D3 34 01 1A 2A 6B 63 B8 7A 1E A7 A1 CE DD E5 5F 41 49 8D 1B 74 5A
4E 9D EF E7 CB 39 31 3E 8B 2E EC F9 37 FB 3B 9B 47 AF DE 86 94 4E 4D FC 30 95 A0 8B B1
3D D1 AF B6 FC 9D DD 2D B3 6F 18 FB B5 60 25 51 45 C9 5E 86 6D 94 FF DB BE 29 9A 1E C2
CC AF F3 51 1C 7E 06 D3 BF 5E FD E1 B2 62 34 62 4D 44 BA A9 25 7A B4 04 2B 7F B9 55 4F
8E 54 DC 32 8D A7 87 88 24 6F 59 C5 30 51 0F 5C 5F 8A 6F

SK1.CBC_decrypt(Data, IV(SSC)) : 7F 49 82 01 09 81 82 01 00 98 A9 D5 3B 93 C1 CB CD 84 4C
5D D2 C4 AB 64 9C 64 9C AA 05 BA 35 C4 BE 50 77 2D 93 2F 9A 29 67 26 1D EF 94 B6 AA B2
F0 B2 91 C0 5E 75 28 99 B6 C5 E9 F8 C1 8D C2 C2 B8 91 6F 71 40 55 76 7C EC BE 84 4D 9B
BC DA 6E 1B 0D C9 C2 04 53 60 98 21 F3 64 B0 31 5F A2 A4 5E 1D FB AE 7C EC 91 2A A0 DC
D8 F7 1D E9 FC 51 22 CC 65 1A 1B EB F7 D7 F4 2A 38 4C 2B 19 3A DD A8 BE A6 41 88 DD 20
E2 32 AC D4 AE E0 A8 00 83 3D AC 38 33 27 BA 9A FD B0 CD EC 7E 38 36 03 76 85 90 5E B6
CF B6 FB F1 6C 8C 11 F2 84 70 F0 60 38 37 80 AD 2D 2E 07 AA B7 3B BA 87 7D F4 83 50 97
0F B4 59 AE 5B D2 84 DD D2 87 63 CB F2 F3 61 49 0C 59 B1 F1 04 B5 32 90 79 92 AA 52 BF
9F A2 21 76 09 D9 B7 E0 68 26 EF 14 D2 97 F1 B0 7D FE 67 D4 B1 4A 91 83 D6 2F CF 4A A3
FC 48 94 6C 17 2D C5 23 CA D0 7F 02 85 D3 82 03 01 00 01 80 00

// Secure command.
CLA = CLA | 0C: 0C
SSC(05 18 09 B6 A2 DF 2D 24) + 1 = 05 18 09 B6 A2 DF 2D 25
// Prepare MACData
Append header: 0C 06 01 01
Append 80000000: 0C 06 01 01 80 00 00 00
MACData = Append ISO9797 method 2 padding:
0C 06 01 01 80 00 00 00 80 00 00 00 00 00 00 00
// Calculate MAC
SK2Key1 = SK2[0..7]: 27 06 B4 CF EF 08 DE 18
SK2Key2 = SK2[8..15]: 8A BF 71 01 45 3F E4 4C
MAC = SK2Key1.CBC_encrypt(MACData, IV(SSC)): 6C 6E F3 C5 85 F2 02 73
MAC = SK2Key2.decrypt(MAC): C3 DF 73 B5 F4 B5 FD CB
MAC = SK2Key1.encrypt(MAC): 98 6D CD 1C 57 30 02 D6
// Wrap MAC into TLV with tag 8E.
MAC = Tag(8E) || Length || Value(MAC): 8E 08 98 6D CD 1C 57 30 02 D6
Data = Data || MAC: 8E 08 98 6D CD 1C 57 30 02 D6
>> 0C 06 01 01 00 00 0A 8E 08 98 6D CD 1C 57 30 02 D6 00 00 - SECURE GENERATE KEY
<< 87 82 01 10 50 61 C7 FB 23 9E 4F 59 B5 30 8B 56 40 9E 25 81 4E F3 96 44 5F EA ED 28 44
9D 9A D3 5C 25 6E 7F 66 84 5E 5F FF 0B F5 64 10 6C 5D 9B 49 09 84 0B D4 B2 91 44 C6 D7
EE D3 A9 7C 8A 21 01 61 2E C8 4A 23 1C BC F9 1D C0 E4 A6 D4 65 1A 6C 3A 1C 88 94 0F 8C
20 ED 38 A8 7C 36 49 A6 8F F3 F3 A8 FD 2A 13 24 C8 AC 6D FB 69 0C 4F 33 6C 84 44 C2 CB
65 FE 39 F3 FF CE 45 3F 68 E7 44 87 6E 21 1E 8D C6 75 13 E5 3F D1 54 D0 9E 8C F1 79 74
87 17 F4 AF C5 D9 F3 8F 50 8F 8C AA 39 BD AD EB 4D A6 25 D7 E4 B4 AF 9F CE 8F 62 26 CA
```



```
90 1E 4A A5 CC 6B CC D8 79 9A 02 CF 54 45 C9 99 29 A5 59 70 E4 51 22 61 72 1B C9 C9 BE
78 FA 63 F0 E9 AA 78 35 45 12 44 AD E9 DC 3A 1C FA 99 86 9A 4D 5E 49 2E 8C 73 4C 87 20
EA 7E AB F5 16 9E 32 A3 6C A0 D1 60 DC B8 CD 2E 7A 2F 49 72 C5 03 46 2A B6 BB B5 59 FB
32 C7 6A D4 31 6A 9D A3 4A BC 98 19 10 16 3F 8E 08 25 AF 1A 3C 91 1A 42 7D 90 00 - OK

// Verify MAC and decrypt
SSC(05 18 09 B6 A2 DF 2D 25) + 1 = 05 18 09 B6 A2 DF 2D 26
// Prepare MACData
Append R-APDU data without MAC TLV(8E):
87 82 01 10 50 61 C7 FB 23 9E 4F 59 B5 30 8B 56 40 9E 25 81 4E F3 96 44 5F
EA ED 28 44 9D 9A D3 5C 25 6E 7F 66 84 5E 5F FF 0B F5 64 10 6C 5D 9B 49 09
84 0B D4 B2 91 44 C6 D7 EE D3 A9 7C 8A 21 01 61 2E C8 4A 23 1C BC F9 1D C0
E4 A6 D4 65 1A 6C 3A 1C 88 94 0F 8C 20 ED 38 A8 7C 36 49 A6 8F F3 F3 A8 FD
2A 13 24 C8 AC 6D FB 69 0C 4F 33 6C 84 44 C2 CB 65 FE 39 F3 FF CE 45 3F 68
E7 44 87 6E 21 1E 8D C6 75 13 E5 3F D1 54 D0 9E 8C F1 79 74 87 17 F4 AF C5
D9 F3 8F 50 8F 8C AA 39 BD AD EB 4D A6 25 D7 E4 B4 AF 9F CE 8F 62 26 CA 90
1E 4A A5 CC 6B CC D8 79 9A 02 CF 54 45 C9 99 29 A5 59 70 E4 51 22 61 72 1B
C9 C9 BE 78 FA 63 F0 E9 AA 78 35 45 12 44 AD E9 DC 3A 1C FA 99 86 9A 4D 5E
49 2E 8C 73 4C 87 20 EA 7E AB F5 16 9E 32 A3 6C A0 D1 60 DC B8 CD 2E 7A 2F
49 72 C5 03 46 2A B6 BB B5 59 FB 32 C7 6A D4 31 6A 9D A3 4A BC 98 19 10 16
3F
MACData = Append ISO9797 method 2 padding:
87 82 01 10 50 61 C7 FB 23 9E 4F 59 B5 30 8B 56 40 9E 25 81 4E F3 96 44 5F
EA ED 28 44 9D 9A D3 5C 25 6E 7F 66 84 5E 5F FF 0B F5 64 10 6C 5D 9B 49 09
84 0B D4 B2 91 44 C6 D7 EE D3 A9 7C 8A 21 01 61 2E C8 4A 23 1C BC F9 1D C0
E4 A6 D4 65 1A 6C 3A 1C 88 94 0F 8C 20 ED 38 A8 7C 36 49 A6 8F F3 F3 A8 FD
2A 13 24 C8 AC 6D FB 69 0C 4F 33 6C 84 44 C2 CB 65 FE 39 F3 FF CE 45 3F 68
E7 44 87 6E 21 1E 8D C6 75 13 E5 3F D1 54 D0 9E 8C F1 79 74 87 17 F4 AF C5
D9 F3 8F 50 8F 8C AA 39 BD AD EB 4D A6 25 D7 E4 B4 AF 9F CE 8F 62 26 CA 90
1E 4A A5 CC 6B CC D8 79 9A 02 CF 54 45 C9 99 29 A5 59 70 E4 51 22 61 72 1B
C9 C9 BE 78 FA 63 F0 E9 AA 78 35 45 12 44 AD E9 DC 3A 1C FA 99 86 9A 4D 5E
49 2E 8C 73 4C 87 20 EA 7E AB F5 16 9E 32 A3 6C A0 D1 60 DC B8 CD 2E 7A 2F
49 72 C5 03 46 2A B6 BB B5 59 FB 32 C7 6A D4 31 6A 9D A3 4A BC 98 19 10 16
3F 80 00 00 00
MAC: 25 AF 1A 3C 91 1A 42 7D
_MAC = SK2Key1.CBC_encrypt(MACData, IV(SSC)): A4 22 14 7F 40 1F FC 91
_MAC = SK2Key2.decrypt(_MAC): CA BF 0F 42 DF D0 B6 9D
_MAC = SK2Key1.encrypt(_MAC): 25 AF 1A 3C 91 1A 42 7D
MAC == _MAC (true)
Unwrap cryptogram from TLV:
50 61 C7 FB 23 9E 4F 59 B5 30 8B 56 40 9E 25 81 4E F3 96 44 5F EA ED 28 44 9D 9A D3 5C
25 6E 7F 66 84 5E 5F FF 0B F5 64 10 6C 5D 9B 49 09 84 0B D4 B2 91 44 C6 D7 EE D3 A9 7C
8A 21 01 61 2E C8 4A 23 1C BC F9 1D C0 E4 A6 D4 65 1A 6C 3A 1C 88 94 0F 8C 20 ED 38 A8
7C 36 49 A6 8F F3 F3 A8 FD 2A 13 24 C8 AC 6D FB 69 0C 4F 33 6C 84 44 C2 CB 65 FE 39 F3
FF CE 45 3F 68 E7 44 87 6E 21 1E 8D C6 75 13 E5 3F D1 54 D0 9E 8C F1 79 74 87 17 F4 AF
C5 D9 F3 8F 50 8F 8C AA 39 BD AD EB 4D A6 25 D7 E4 B4 AF 9F CE 8F 62 26 CA 90 1E 4A A5
CC 6B CC D8 79 9A 02 CF 54 45 C9 99 29 A5 59 70 E4 51 22 61 72 1B C9 C9 BE 78 FA 63 F0
E9 AA 78 35 45 12 44 AD E9 DC 3A 1C FA 99 86 9A 4D 5E 49 2E 8C 73 4C 87 20 EA 7E AB F5
16 9E 32 A3 6C A0 D1 60 DC B8 CD 2E 7A 2F 49 72 C5 03 46 2A B6 BB B5 59 FB 32 C7 6A D4
31 6A 9D A3 4A BC 98 19 10 16 3F
SK1.CBC_decrypt(Data, IV(SSC)):
7F 49 82 01 09 81 82 01 00 9B 1A 9D 61 32 D8 59 3B 6F 71 2D 5F 6E CE 19 D8 53 CA 5C CE
A3 20 79 E7 0D EF 60 A5 E6 C6 8B EC 4C AD B8 35 13 D0 E4 22 52 E9 C6 54 AD CB AA 23 14
F7 28 54 E3 8C 5E 0F 2F 63 9B 52 6A FA C5 6C 71 EB F5 D6 00 59 5A 6F 85 75 76 3B 83 9A
FB F7 7D 7E FC A8 46 DA 83 A8 92 06 28 21 A0 C4 7F 70 AB 7F 89 39 2F 48 52 1F F4 88 F8
DD 7F 30 B1 2A 25 B5 99 0A A6 FF 8C 48 1F A2 6A E3 D0 30 61 4E BE E3 6D 2D 67 E2 CF A1
71 88 32 E0 A5 88 2D 34 0C 7B 8C 0B 0B C7 04 CA A7 6E 83 12 93 F7 BA 06 D0 2B 56 4D DB
14 59 E1 8A E1 06 54 7A BB 47 06 AA 06 EE 55 68 85 AE 1A CE 19 A2 A9 7F C3 A2 C8 C0 83
0D EE 45 6F 8B 81 1F 95 40 E5 B8 3A 60 E2 2B 53 2A 48 EE 03 C7 1D 6D 06 F7 FB 6B 76 74
01 2D E3 B3 47 0A 11 B1 2C 75 35 B1 55 4F CB 75 22 64 93 53 D4 E9 AF 30 CD EA C1 A4 82
9D EE C1 D5 82 03 01 00 01 80 00
```

## Replace Certificates

### Replacing of Authentication Certificate

// Calculate cardholder CMK. "47101010033" as seed, from EF 5044 record 7.



## TB-SPEC-EstEID-Chip-App-v3.5-20170314 Politsei- ja Piirivalveamet

```
SHA1("47101010033") = 74 B5 97 30 6B C5 9B 67 2F B1 64 B2 0F 36 23 3A B3 35 37 0C
Take 16 leftmost bytes of calculated SHA1:    74 B5 97 30 6B C5 9B 67 2F B1 64 B2 0F 36
                                              23 3A
CMK_CERT.encrypt(74 B5 97 30 6B C5 9B 67 2F B1 64 B2 0F 36 23 3A) =
                                              BA F8 F0 00 7A 4E 9A 38
                                              46 38 46 24 6C FE 88 B4

Set off LSB bits of every byte: BAF8F0007A4E9A38463846246CFE88B4 (cardholder CMK_CERT)


// Verify PIN1
A>> T=1 (4+0004) 00200001 04 31323334
A<< (0000+2) (110ms) 9000


// Mutual Authenticate with cardholder CMK_CERT
A>> T=1 (4+0000) 00840000 00
A<< (0008+2) (20ms) 33221f7bffb5dd79 9000
RND.ICC: 33221f7bffb5dd79
RND.IFD: 2f80db2b5d158d52
Payload: 2f80db2b5d158d5233221f7bffb5dd795f2a26a6ede3b00ba55a77784b09fe75131c679
bde339afcdab5a4fdd39da9ed
Crypted: 66f0578da0dd4085314439dd4fab4e27edb898aa57ce67a1705ab4d9a109c557c733565
58288608b740fb05caa8995f6
A>> T=1 (4+0048) 00820003 30 66f0578da0dd4085314439dd4fab4e27edb898aa57ce67a1705
ab4d9a109c557c73356558288608b740fb05caa8995f630
A<< (0048+2) (170ms) 209dd390b9e3a7831958ae278c32b9a977ed8dd157243a737db24ac7526
daea80294a0c71051168ec68b058010d043dd 9000
Encrypted: 209dd390b9e3a7831958ae278c32b9a977ed8dd157243a737db24ac7526daea80294a
0c71051168ec68b058010d043dd
Decrypted: 33221f7bffb5dd792f80db2b5d158d52d38ff9996469bec19ef19fdb57d04bf5b5c72
47e8dce07316a743b5ac077eb3d
K.ICC: d38ff9996469bec19ef19fdb57d04bf5b5c7247e8dce07316a743b5ac077eb3d
K.IFD: 5f2a26a6ede3b00ba55a77784b09fe75131c679bde339afcdab5a4fdd39da9ed
K.XOR: 8ca5df3f898a0eca3babe8a31cd9b580a6db43e553fd9dcdb0c19fa713ea42d0
SK1: 8ca5df3f898a0eca3babe8a31cd9b580
SK1: 8ca4df3e898a0ecb3babe9a21cd9b580
SK2: a6db43e553fd9dcdb0c19fa713ea42d0
SK2: a7da43e552fd9dcdb0c19ea713ea43d0
SSC: 5d158d52ffb5dd79
## Mutual Authentication successful ##


Original APDU: 8c0700006f308205423082042aa00302010202100de942ac3831061f5609395bc
7defffd1300d06092a864886f70d01010b0500306c310b30090603550406130245453122302006035
5040a0c19415320536572746966697473656572696d69736b65733b7573311f301d06035504030c1
6544553
APDU payload: 308205423082042aa00302010202100de942ac3831061f5609395bc7defffd1300
d06092a864886f70d01010b0500306c310b300906035504061302454531223020060355040a0c194
15320536572746966697473656572696d69736b65733b7573311f301d06035504030c16544553
Crypt payload: 877101bbad28d0669e99806e76878d7dedf8b66ad2e32d7d07ca2066dff26f0ce
972bca8f29ef55593bbf4a9ca6816belc3a31b009c172f4f889fd3f8ba6da43ea080faba53c6ac95
e152f4b2b5cd4f5069d73982c2eae72450166bb20f635a5638471aced771c6a11da4195b2a3560c1
e16f0
Verified APDU: 308205423082042aa00302010202100de942ac3831061f5609395bc7defffd1300
```



---

d06092a864886f70d01010b0500306c310b300906035504061302454531223020060355040a0c194  
15320536572746966697473656572696d69736b657336b7573311f301d06035504030c1654455380  
SSC+1: 5d158d52ffb5dd7a  
MAC payload: 123 8c07000080000000877101bbad28d0669e99806e76878d7dedf8b66ad2e32d  
7d07ca2066dff26f0ce972bca8f29ef55593bbf4a9ca6816be1c3a31b009c172f4f889fd3f8ba6da  
43ea080faba53c6ac95e152f4b2b5cd4f5069d73982c2eae72450166bb20f635a5638471aced771c  
6a11da4195b2a3560c1e16f0  
MAC: 77efd6a7671e8cf1  
Final APDU: 8c0700007d877101bbad28d0669e99806e76878d7dedf8b66ad2e32d7d07ca2066df  
f26f0ce972bca8f29ef55593bbf4a9ca6816be1c3a31b009c172f4f889fd3f8ba6da43ea080faba5  
3c6ac95e152f4b2b5cd4f5069d73982c2eae72450166bb20f635a5638471aced771c6a11da4195b2  
a3560c1e16f08e0877efd6a7671e8cf100  
A>> T=1 (4+0125) 8c070000 7d 877101bbad28d0669e99806e76878d7dedf8b66ad2e32d7d07c  
a2066dff26f0ce972bca8f29ef55593bbf4a9ca6816be1c3a31b009c172f4f889fd3f8ba6da43ea0  
80faba53c6ac95e152f4b2b5cd4f5069d73982c2eae72450166bb20f635a5638471aced771c6a11d  
a4195b2a3560c1e16f08e0877efd6a7671e8cf100  
A<< (0014+2) (180ms) 990290008e08f727841ca2f9ee95 9000  
Card MAC: f727841ca2f9ee95  
Response MAC payload: 4 99029000  
Response MAC: f727841ca2f9ee95  
ResponseAPDU: 9000  
Original APDU: 8c07006f6f54206f66204553544549442d534b20323031313118301606092a864  
886f70d0109011609706b6940736b2e6565301e170d3135303932383132353830335a170d3230303  
932323230353935395a30819b310b3009060355040613024545310f300d060355040a0c064553544  
5494431  
APDU payload: 54206f66204553544549442d534b20323031313118301606092a864886f70d010  
9011609706b6940736b2e6565301e170d3135303932383132353830335a170d32303039323232303  
53935395a30819b310b3009060355040613024545310f300d060355040a0c0645535445494431  
Crypt payload: 877101ff91668d8da5636180a8ef90bf98d8f7644108249fc6f7bf0fdb6dd4070  
97cb2d7bcd7dd1e7b52a41b4885bae3c9086137fd183dc881744b23f30f20e227180d134d9d08662  
2566f028a8a9e9f74712ad54b790645ee6457a25efbe7dc6b60b893dd3b1b79923fe1053b7ce8099  
3dc67  
Verified APDU: 54206f66204553544549442d534b20323031313118301606092a864886f70d010  
9011609706b6940736b2e6565301e170d3135303932383132353830335a170d32303039323232303  
53935395a30819b310b3009060355040613024545310f300d060355040a0c064553544549443180  
SSC+1: 5d158d52ffb5dd7c  
MAC payload: 123 8c07006f80000000877101ff91668d8da5636180a8ef90bf98d8f764410824  
9fc6f7bf0fdb6dd407097cb2d7bcd7dd1e7b52a41b4885bae3c9086137fd183dc881744b23f30f20  
e227180d134d9d086622566f028a8a9e9f74712ad54b790645ee6457a25efbe7dc6b60b893dd3b1b  
79923fe1053b7ce80993dc67  
MAC: cf3ba396ff745e74  
Final APDU: 8c07006f7d877101ff91668d8da5636180a8ef90bf98d8f7644108249fc6f7bf0fdb  
6dd407097cb2d7bcd7dd1e7b52a41b4885bae3c9086137fd183dc881744b23f30f20e227180d134d  
9d086622566f028a8a9e9f74712ad54b790645ee6457a25efbe7dc6b60b893dd3b1b79923fe1053b  
7ce80993dc678e08cf3ba396ff745e7400  
A>> T=1 (4+0125) 8c07006f 7d 877101ff91668d8da5636180a8ef90bf98d8f7644108249fc6f  
7bf0fdb6dd407097cb2d7bcd7dd1e7b52a41b4885bae3c9086137fd183dc881744b23f30f20e2271  
80d134d9d086622566f028a8a9e9f74712ad54b790645ee6457a25efbe7dc6b60b893dd3b1b79923  
fe1053b7ce80993dc678e08cf3ba396ff745e7400  
A<< (0014+2) (130ms) 990290008e08fbb9ad38d206220 9000  
Card MAC: 6fbb9ad38d206220  
Response MAC payload: 4 99029000





## TB-SPEC-EstEID-Chip-App-v3.5-20170314 Politsei- ja Piirivalveamet

Response MAC: 6fbb9ad38d206220  
ResponseAPDU: 9000  
Original APDU: 8c0700de6f173015060355040b0c0e61757468656e7469636174696f6e3126302  
406035504030c1d4dc3844e4e494b2c4d4152492d4c4949532c34373130313031303033333110300  
e06035504040c074dc3844e4e494b31123010060355042a0c094d4152492d4c49495331143012060  
3550405  
APDU payload: 173015060355040b0c0e61757468656e7469636174696f6e31263024060355040  
30c1d4dc3844e4e494b2c4d4152492d4c4949532c34373130313031303033333110300e060355040  
40c074dc3844e4e494b31123010060355042a0c094d4152492d4c494953311430120603550405  
Crypt payload: 877101abffcdfe76d0172712f2fb1bdb336479bcae60782b8be582b1d9d0eea89  
1f1f5cacd69ef29741cca734ebab2983bf92b74283f2668887b6610713a86ce04be70b8ec805e949  
10432da29e1ce41afc002d13101a2a321eded1d49428ac0aaf3fff9ede243a235c890156eef74d5c  
f2e29  
Verified APDU: 173015060355040b0c0e61757468656e7469636174696f6e31263024060355040  
30c1d4dc3844e4e494b2c4d4152492d4c4949532c34373130313031303033333110300e060355040  
40c074dc3844e4e494b31123010060355042a0c094d4152492d4c49495331143012060355040580  
SSC+1: 5d158d52ffb5dd7e  
MAC payload: 123 8c0700de80000000877101abffcdfe76d0172712f2fb1bdb336479bcae6078  
2b8be582b1d9d0eea891f1f5cacd69ef29741cca734ebab2983bf92b74283f2668887b6610713a86  
ce04be70b8ec805e94910432da29e1ce41afc002d13101a2a321eded1d49428ac0aaf3fff9ede243  
a235c890156eef74d5cf2e29  
MAC: d05f4447c1d12bb1  
Final APDU: 8c0700de7d877101abffcdfe76d0172712f2fb1bdb336479bcae60782b8be582b1d9  
d0eea891f1f5cacd69ef29741cca734ebab2983bf92b74283f2668887b6610713a86ce04be70b8ec  
805e94910432da29e1ce41afc002d13101a2a321eded1d49428ac0aaf3fff9ede243a235c890156e  
ef74d5cf2e298e08d05f4447c1d12bb100  
A>> T=1 (4+0125) 8c0700de 7d 877101abffcdfe76d0172712f2fb1bdb336479bcae60782b8be  
582b1d9d0eea891f1f5cacd69ef29741cca734ebab2983bf92b74283f2668887b6610713a86ce04b  
e70b8ec805e94910432da29e1ce41afc002d13101a2a321eded1d49428ac0aaf3fff9ede243a235c  
890156eef74d5cf2e298e08d05f4447c1d12bb100  
A<< (0014+2) (130ms) 990290008e0832acf6f7ffa114cd 9000  
Card MAC: 32acf6f7ffa114cd  
Response MAC payload: 4 99029000  
Response MAC: 32acf6f7ffa114cd  
ResponseAPDU: 9000  
Original APDU: 8c07014d6f130b343731303130313030333330820122300d06092a864886f70d0  
1010105000382010f003082010a0282010100959fe0c2e8fde7f9111cbf74377bcac7dca81f65c1d  
fe549be57eed684486f45aa3bfa5b83862b9c3f55ff01a00c340f21ac5f86519e80a5688f5d8f309  
1cb0344  
APDU payload: 130b343731303130313030333330820122300d06092a864886f70d01010105000  
382010f003082010a0282010100959fe0c2e8fde7f9111cbf74377bcac7dca81f65c1dfe549be57e  
ed684486f45aa3bfa5b83862b9c3f55ff01a00c340f21ac5f86519e80a5688f5d8f3091cb0344  
Crypt payload: 87710121b3e8afc12164d170ddc0d0f94701c82941de7ce13c151ff383a7979fc  
5feee4893a5376f2d07a5fcb7bf783218006086cd96e44bfe6ce4941771c9fab77a1f41f3901b1df  
cf549866f8c6b7c16b0e398a77f299627d7d1fdc86c910c10d490c43876724d3f2d09cad46623dfe  
1454a  
Verified APDU: 130b343731303130313030333330820122300d06092a864886f70d01010105000  
382010f003082010a0282010100959fe0c2e8fde7f9111cbf74377bcac7dca81f65c1dfe549be57e  
ed684486f45aa3bfa5b83862b9c3f55ff01a00c340f21ac5f86519e80a5688f5d8f3091cb034480  
SSC+1: 5d158d52ffb5dd80  
MAC payload: 123 8c07014d8000000087710121b3e8afc12164d170ddc0d0f94701c82941de7c  
e13c151ff383a7979fc5feee4893a5376f2d07a5fcb7bf783218006086cd96e44bfe6ce4941771c9



fab77a1f41f3901b1dfcf549866f8c6b7c16b0e398a77f299627d7d1fdc86c910c10d490c4387672  
4d3f2d09cad46623dfe1454a  
MAC: 1bc27cd0b270f481  
Final APDU: 8c07014d7d87710121b3e8afc12164d170ddc0d0f94701c82941de7ce13c151ff383  
a7979fc5feee4893a5376f2d07a5fcb7bf783218006086cd96e44bfe6ce4941771c9fab77a1f41f3  
901b1dfcf549866f8c6b7c16b0e398a77f299627d7d1fdc86c910c10d490c43876724d3f2d09cad4  
6623dfe1454a8e081bc27cd0b270f48100  
A>> T=1 (4+0125) 8c07014d 7d 87710121b3e8afc12164d170ddc0d0f94701c82941de7ce13c1  
51ff383a7979fc5feee4893a5376f2d07a5fcb7bf783218006086cd96e44bfe6ce4941771c9fab77  
a1f41f3901b1dfcf549866f8c6b7c16b0e398a77f299627d7d1fdc86c910c10d490c43876724d3f2  
d09cad46623dfe1454a8e081bc27cd0b270f48100  
A<< (0014+2) (127ms) 990290008e08b452412c84c1aa45 9000  
Card MAC: b452412c84c1aa45  
Response MAC payload: 4 99029000  
Response MAC: b452412c84c1aa45  
ResponseAPDU: 9000  
Original APDU: 8c0701bc6fffb3bce3616701d0328795e274af46f3c9a2c71384e4114eda5ae174  
06f1f0d7f620fb7f3f5cf8f49dde9155f4837509d9722379b7c592fedcf464d78f194c85412f7ea0  
f8786481cb7623b2e7e58c8a71e8af8455e875fc6816523ad77526185b9f6230bdf9da73fc0f170  
67ebefe  
APDU payload: fb3bce3616701d0328795e274af46f3c9a2c71384e4114eda5ae17406f1f0d7f6  
20fb7f3f5cf8f49dde9155f4837509d9722379b7c592fedcf464d78f194c85412f7ea0f8786481cb  
7623b2e7e58c8a71e8af8455e875fc6816523ad77526185b9f6230bdf9da73fc0f17067ebefe  
Crypt payload: 877101faa3b9876baaa8035bc4ce6c2c6ffd8b559de0f0fe030d44c34b9a2ff90  
f434ca8f76bcafb8ab5932b11fc47d2133648a65c9f3465f1e0aed0d9ccb66c988e747e5263a314f  
e74f3abe60cf717323fc90c6de262265a07ac172a150efa45ecf3b61d640942f96259712eef4fff2  
bcd09  
Verified APDU: fb3bce3616701d0328795e274af46f3c9a2c71384e4114eda5ae17406f1f0d7f6  
20fb7f3f5cf8f49dde9155f4837509d9722379b7c592fedcf464d78f194c85412f7ea0f8786481cb  
7623b2e7e58c8a71e8af8455e875fc6816523ad77526185b9f6230bdf9da73fc0f17067ebefe80  
SSC+1: 5d158d52ffb5dd82  
MAC payload: 123 8c0701bc80000000877101faa3b9876baaa8035bc4ce6c2c6ffd8b559de0f0  
fe030d44c34b9a2ff90f434ca8f76bcafb8ab5932b11fc47d2133648a65c9f3465f1e0aed0d9ccb6  
6c988e747e5263a314fe74f3abe60cf717323fc90c6de262265a07ac172a150efa45ecf3b61d6409  
42f96259712eef4fff2bcd09  
MAC: ad8f24caf566847b  
Final APDU: 8c0701bc7d877101faa3b9876baaa8035bc4ce6c2c6ffd8b559de0f0fe030d44c34b  
9a2ff90f434ca8f76bcafb8ab5932b11fc47d2133648a65c9f3465f1e0aed0d9ccb66c988e747e52  
63a314fe74f3abe60cf717323fc90c6de262265a07ac172a150efa45ecf3b61d640942f96259712e  
ef4fff2bcd098e08ad8f24caf566847b00  
A>> T=1 (4+0125) 8c0701bc 7d 877101faa3b9876baaa8035bc4ce6c2c6ffd8b559de0f0fe030  
d44c34b9a2ff90f434ca8f76bcafb8ab5932b11fc47d2133648a65c9f3465f1e0aed0d9ccb66c988  
e747e5263a314fe74f3abe60cf717323fc90c6de262265a07ac172a150efa45ecf3b61d640942f96  
259712eef4fff2bcd098e08ad8f24caf566847b00  
A<< (0014+2) (130ms) 990290008e08e35aa8dd9102dada 9000  
Card MAC: e35aa8dd9102dada  
Response MAC payload: 4 99029000  
Response MAC: e35aa8dd9102dada  
ResponseAPDU: 9000  
Original APDU: 8c07022b6fb06ab5151720856dfa7451567811b5762ee2e64d283af30232c070f  
326058eb56f238bcc77437b5021f7203132c7681876052f0d5658fe446514856dee5ab49c9880364  
24fca100cf842157e4f9538c10203012001a38201ae308201aa30090603551d1304023000300e060



3551d0f

APDU payload: b06ab5151720856dfa7451567811b5762ee2e64d283af30232c070f326058eb56f238bcc77437b5021f7203132c7681876052f0d5658fe446514856dee5ab49c988036424fca100cf842157e4f9538c10203012001a38201ae308201aa30090603551d1304023000300e0603551d0f

Crypt payload: 877101ab69e00c3b9f80451b38afbbbee020c7197070c48cd921938afa74e7c199ae27b63b013f77b39868de983441c62e5c46d0028a3aa19bb7eec43d24e1590b8bd5362831afaab67029101c3455337ec92af17741885feff21d03554874ee33377bd00eda2a2e687426afb1f8a3f2c39b0f

Verified APDU: b06ab5151720856dfa7451567811b5762ee2e64d283af30232c070f326058eb56f238bcc77437b5021f7203132c7681876052f0d5658fe446514856dee5ab49c988036424fca100cf842157e4f9538c10203012001a38201ae308201aa30090603551d1304023000300e0603551d0f80

SSC+1: 5d158d52ffb5dd84

MAC payload: 123 8c07022b80000000877101ab69e00c3b9f80451b38afbbbee020c7197070c48cd921938afa74e7c199ae27b63b013f77b39868de983441c62e5c46d0028a3aa19bb7eec43d24e1590b8bd5362831afaab67029101c3455337ec92af17741885feff21d03554874ee33377bd00eda2a2e687426afb1f8a3f2c39b0f

MAC: 6750187cb083c9e9

Final APDU: 8c07022b7d877101ab69e00c3b9f80451b38afbbbee020c7197070c48cd921938afa74e7c199ae27b63b013f77b39868de983441c62e5c46d0028a3aa19bb7eec43d24e1590b8bd5362831afaab67029101c3455337ec92af17741885feff21d03554874ee33377bd00eda2a2e687426afb1f8a3f2c39b0f8e086750187cb083c9e900

A>> T=1 (4+0125) 8c07022b 7d 877101ab69e00c3b9f80451b38afbbbee020c7197070c48cd921938afa74e7c199ae27b63b013f77b39868de983441c62e5c46d0028a3aa19bb7eec43d24e1590b8bd5362831afaab67029101c3455337ec92af17741885feff21d03554874ee33377bd00eda2a2e687426afb1f8a3f2c39b0f8e086750187cb083c9e900

A<< (0014+2) (130ms) 990290008e081fc6689be6ed81c8 9000

Card MAC: 1fc6689be6ed81c8

Response MAC payload: 4 99029000

Response MAC: 1fc6689be6ed81c8

ResponseAPDU: 9000

Original APDU: 8c07029a6f0101ff0404030204b03081990603551d2004819130818e30818b060a2b06010401ce1f030101307d305806082b06010505070202304c1e4a00410069006e0075006c0074002000740065007300740069006d006900730065006b0073002e0020004f006e006c007900200066006f00

APDU payload: 0101ff0404030204b03081990603551d2004819130818e30818b060a2b06010401ce1f030101307d305806082b06010505070202304c1e4a00410069006e0075006c0074002000740065007300740069006d006900730065006b0073002e0020004f006e006c007900200066006f00

Crypt payload: 87710160e8450cd63e1eb033c9bf3f2fba91082723ec4810087fe8e6515e2aac301a2be2cc012abb54e4207e88fa801a94914d469c241fbda428efc819d3420f320649b10f4c140369a7a9dd500133763850339fcc541da93ba4da0185a9a6baef296deb836531da6e471479edc11af7c1779

Verified APDU: 0101ff0404030204b03081990603551d2004819130818e30818b060a2b06010401ce1f030101307d305806082b06010505070202304c1e4a00410069006e0075006c0074002000740065007300740069006d006900730065006b0073002e0020004f006e006c007900200066006f0080

SSC+1: 5d158d52ffb5dd86

MAC payload: 123 8c07029a8000000087710160e8450cd63e1eb033c9bf3f2fba91082723ec4810087fe8e6515e2aac301a2be2cc012abb54e4207e88fa801a94914d469c241fbda428efc819d3420f320649b10f4c140369a7a9dd500133763850339fcc541da93ba4da0185a9a6baef296deb836531da6e471479edc11af7c1779

MAC: f025942cbc9ce6df

Final APDU: 8c07029a7d87710160e8450cd63e1eb033c9bf3f2fba91082723ec4810087fe8e6515e2aac301a2be2cc012abb54e4207e88fa801a94914d469c241fbda428efc819d3420f320649b10



```
f4c140369a7a9dd500133763850339fcc541da93ba4da0185a9a6baef296deb836531da6e471479e
dc11af7c17798e08f025942cbc9ce6df00
A>> T=1 (4+0125) 8c07029a 7d 87710160e8450cd63e1eb033c9bf3f2fba91082723ec4810087
fe8e6515e2aac301a2be2cc012abb54e4207e88fa801a94914d469c241fbeda428efc819d3420f32
0649b10f4c140369a7a9dd500133763850339fcc541da93ba4da0185a9a6baef296deb836531da6e
471479edc11af7c17798e08f025942cbc9ce6df00
A<< (0014+2) (130ms) 990290008e0825a424ae0b13ff40 9000
Card MAC: 25a424ae0b13ff40
Response MAC payload: 4 99029000
Response MAC: 25a424ae0b13ff40
ResponseAPDU: 9000
Original APDU: 8c0703096f72002000740065007300740069006e0067002e302106082b0601050
50702011615687474703a2f2f7777772e736b2e65652f6370732f30240603551d11041d301b81196
d6172692d6c6969732e6d616e6e696b4065657374692e6565301d0603551d0e041604148858537de
11db204
APDU payload: 72002000740065007300740069006e0067002e302106082b06010505070201161
5687474703a2f2f7777772e736b2e65652f6370732f30240603551d11041d301b81196d6172692d6
c6969732e6d616e6e696b4065657374692e6565301d0603551d0e041604148858537de11db204
Crypt payload: 8771017c54b841eb3fbe18679e6500e8176200ded0a1390795c286250a0b8dfe9
78260798d07c01704f2b1b2d84c328c5993bdfcfc5420010888dfa0d68fe687c26e94cdd5d5f4d80
58197cff911129c9934c8c48e8c1367abaae5e455863c972d6b04fb64742382bc640da0dfb43e29c
45e5f
Verified APDU: 72002000740065007300740069006e0067002e302106082b06010505070201161
5687474703a2f2f7777772e736b2e65652f6370732f30240603551d11041d301b81196d6172692d6
c6969732e6d616e6e696b4065657374692e6565301d0603551d0e041604148858537de11db20480
SSC+1: 5d158d52ffb5dd88
MAC payload: 123 8c070309800000008771017c54b841eb3fbe18679e6500e8176200ded0a139
0795c286250a0b8dfe978260798d07c01704f2b1b2d84c328c5993bdfcfc5420010888dfa0d68fe6
87c26e94cdd5d5f4d8058197cff911129c9934c8c48e8c1367abaae5e455863c972d6b04fb647423
82bc640da0dfb43e29c45e5f
MAC: 8e29856db3dfd1d4
Final APDU: 8c0703097d8771017c54b841eb3fbe18679e6500e8176200ded0a1390795c286250a
0b8dfe978260798d07c01704f2b1b2d84c328c5993bdfcfc5420010888dfa0d68fe687c26e94cdd5
d5f4d8058197cff911129c9934c8c48e8c1367abaae5e455863c972d6b04fb64742382bc640da0df
b43e29c45e5f8e088e29856db3dfd1d400
A>> T=1 (4+0125) 8c070309 7d 8771017c54b841eb3fbe18679e6500e8176200ded0a1390795c
286250a0b8dfe978260798d07c01704f2b1b2d84c328c5993bdfcfc5420010888dfa0d68fe687c26
e94cdd5d5f4d8058197cff911129c9934c8c48e8c1367abaae5e455863c972d6b04fb64742382bc6
40da0dfb43e29c45e5f8e088e29856db3dfd1d400
A<< (0014+2) (130ms) 990290008e088521a03bedc6652c 9000
Card MAC: 8521a03bedc6652c
Response MAC payload: 4 99029000
Response MAC: 8521a03bedc6652c
ResponseAPDU: 9000
Original APDU: 8c0703786f9cb384c7c2101ba50e14896930200603551d250101ff04163014060
82b0601050507030206082b06010505070304302206082b060105050701030416301430080606040
08e4601013008060604008e460104301f0603551d2304183016801441b6fec5b1b1b453138cfafa6
2d0346d
APDU payload: 9cb384c7c2101ba50e14896930200603551d250101ff0416301406082b0601050
507030206082b06010505070304302206082b06010505070103041630143008060604008e4601013
008060604008e460104301f0603551d2304183016801441b6fec5b1b1b453138cfafa62d0346d
Crypt payload: 877101de79e55062effda1569492ecc8df202069cd8a44bb5801a432790efa8f3
```



---

88352bb01cafee4dd1d3a700bbaa7d5323280f5613688f6cc781b2f72b6628ee58fcd85ef880b3eb  
9200ff7bf647dabb20ccb5876608ba6e3fcdf269b2dd52c12e48897890c3e8a010f488246afec0ee  
39ee8

Verified APDU: 9cb384c7c2101ba50e14896930200603551d250101ff0416301406082b0601050  
507030206082b06010505070304302206082b06010505070103041630143008060604008e4601013  
008060604008e460104301f0603551d2304183016801441b6fec5b1b1b453138cfafa62d0346d80  
SSC+1: 5d158d52ffb5dd8a

MAC payload: 123 8c0703788000000877101de79e55062effda1569492ecc8df202069cd8a44  
bb5801a432790efa8f388352bb01cafee4dd1d3a700bbaa7d5323280f5613688f6cc781b2f72b662  
8ee58fcd85ef880b3eb9200ff7bf647dabb20ccb5876608ba6e3fcdf269b2dd52c12e48897890c3e  
8a010f488246afec0ee39ee8

MAC: 3b105d67256693c8

Final APDU: 8c0703787d877101de79e55062effda1569492ecc8df202069cd8a44bb5801a43279  
0efa8f388352bb01cafee4dd1d3a700bbaa7d5323280f5613688f6cc781b2f72b6628ee58fcd85ef  
880b3eb9200ff7bf647dabb20ccb5876608ba6e3fcdf269b2dd52c12e48897890c3e8a010f488246  
afec0ee39ee88e083b105d67256693c800

A>> T=1 (4+0125) 8c070378 7d 877101de79e55062effda1569492ecc8df202069cd8a44bb580  
1a432790efa8f388352bb01cafee4dd1d3a700bbaa7d5323280f5613688f6cc781b2f72b6628ee58  
fcd85ef880b3eb9200ff7bf647dabb20ccb5876608ba6e3fcdf269b2dd52c12e48897890c3e8a010  
f488246afec0ee39ee88e083b105d67256693c800

A<< (0014+2) (130ms) 990290008e082b102c68fcdfd723 9000

Card MAC: 2b102c68fcdfd723

Response MAC payload: 4 99029000

Response MAC: 2b102c68fcdfd723

ResponseAPDU: 9000

Original APDU: 8c0703e76f6d22340a30450603551d1f043e303c303aa038a0368634687474703  
a2f2f777772e736b2e65652f7265706f7369746f72792f63726c732f746573745f6573746569643  
23031312e63726c300d06092a864886f70d01010b05000382010100aa50bd1d95ff96557898973fd  
b323aff

APDU payload: 6d22340a30450603551d1f043e303c303aa038a0368634687474703a2f2f7777  
72e736b2e65652f7265706f7369746f72792f63726c732f746573745f657374656964323031312e6  
3726c300d06092a864886f70d01010b05000382010100aa50bd1d95ff96557898973fdb323aff

Crypt payload: 877101cfcbc5c774c2e89ecf9ad64788b33c8bdc53e2ad084bf1ca71150bdcc33  
4540dc07f91e89faa2e9b277e2cbd1e8d5626fe2a72f8871f01d7895e6b0cddf1fecec1fbb936e9d  
de451f13a9b7278ed9ce5c210efelae47b6908e4b7abaf4d19803df012660845a139def2eb78dbdf  
89794

Verified APDU: 6d22340a30450603551d1f043e303c303aa038a0368634687474703a2f2f7777  
72e736b2e65652f7265706f7369746f72792f63726c732f746573745f657374656964323031312e6  
3726c300d06092a864886f70d01010b05000382010100aa50bd1d95ff96557898973fdb323aff80

SSC+1: 5d158d52ffb5dd8c

MAC payload: 123 8c0703e78000000877101cfcbc5c774c2e89ecf9ad64788b33c8bdc53e2ad  
084bf1ca71150bdcc334540dc07f91e89faa2e9b277e2cbd1e8d5626fe2a72f8871f01d7895e6b0c  
ddf1fecec1fbb936e9dde451f13a9b7278ed9ce5c210efelae47b6908e4b7abaf4d19803df012660  
845a139def2eb78dbdf89794

MAC: 83bd8714a8fa86a2

Final APDU: 8c0703e77d877101cfcbc5c774c2e89ecf9ad64788b33c8bdc53e2ad084bf1ca7115  
0bdcc334540dc07f91e89faa2e9b277e2cbd1e8d5626fe2a72f8871f01d7895e6b0cddf1fecec1fb  
b936e9dde451f13a9b7278ed9ce5c210efelae47b6908e4b7abaf4d19803df012660845a139def2e  
b78dbdf897948e0883bd8714a8fa86a200

A>> T=1 (4+0125) 8c0703e7 7d 877101cfcbc5c774c2e89ecf9ad64788b33c8bdc53e2ad084bf  
1ca71150bdcc334540dc07f91e89faa2e9b277e2cbd1e8d5626fe2a72f8871f01d7895e6b0cddf1f  
ecec1fbb936e9dde451f13a9b7278ed9ce5c210efelae47b6908e4b7abaf4d19803df012660845a1



39def2eb78dbdf897948e0883bd8714a8fa86a200  
A<< (0014+2) (130ms) 990290008e0835e924a07a8cf635 9000  
Card MAC: 35e924a07a8cf635  
Response MAC payload: 4 99029000  
Response MAC: 35e924a07a8cf635  
ResponseAPDU: 9000  
Original APDU: 8c0704566f6d0603e0977f6fb4d20e2a3c1829ead671969816a5b4ea37007ab43  
33a51e6813d4268a0f16f1f2d9745d6a3b914c304c82579d2acc673246fdd5fa00c6b2883de28240  
e394c47948826523d625fcae5e633316597b3bb568c9523a7c2417839eb52eeda4c0572fc0c18c1  
432dc33  
APDU payload: 6d0603e0977f6fb4d20e2a3c1829ead671969816a5b4ea37007ab4333a51e6813  
d4268a0f16f1f2d9745d6a3b914c304c82579d2acc673246fdd5fa00c6b2883de28240e394c47948  
826523d625fcae5e633316597b3bb568c9523a7c2417839eb52eeda4c0572fc0c18c1432dc33  
Crypt payload: 877101da3063c134af994a1482c90415968c467e1169d457dbab55af1682e6095  
f9f1f37bc60c11bc40d20c9c125be25fd5662f2625a6fb69dfd3c00026c52747fd784608be01cb3d  
e73e9af07df7fe8fc0d153c323a044404dd5d348bd9f55848c639c949fa5b4883b8e12c354939cd4  
a0754  
Verified APDU: 6d0603e0977f6fb4d20e2a3c1829ead671969816a5b4ea37007ab4333a51e6813  
d4268a0f16f1f2d9745d6a3b914c304c82579d2acc673246fdd5fa00c6b2883de28240e394c47948  
826523d625fcae5e633316597b3bb568c9523a7c2417839eb52eeda4c0572fc0c18c1432dc3380  
SSC+1: 5d158d52ffb5dd8e  
MAC payload: 123 8c0704568000000877101da3063c134af994a1482c90415968c467e1169d4  
57dbab55af1682e6095f9f1f37bc60c11bc40d20c9c125be25fd5662f2625a6fb69dfd3c00026c52  
747fd784608be01cb3de73e9af07df7fe8fc0d153c323a044404dd5d348bd9f55848c639c949fa5b  
4883b8e12c354939cd4a0754  
MAC: 07e9979eb7a3bbbed  
Final APDU: 8c0704567d877101da3063c134af994a1482c90415968c467e1169d457dbab55af16  
82e6095f9f1f37bc60c11bc40d20c9c125be25fd5662f2625a6fb69dfd3c00026c52747fd784608b  
e01cb3de73e9af07df7fe8fc0d153c323a044404dd5d348bd9f55848c639c949fa5b4883b8e12c35  
4939cd4a07548e0807e9979eb7a3bbbed00  
A>> T=1 (4+0125) 8c070456 7d 877101da3063c134af994a1482c90415968c467e1169d457dba  
b55af1682e6095f9f1f37bc60c11bc40d20c9c125be25fd5662f2625a6fb69dfd3c00026c52747fd  
784608be01cb3de73e9af07df7fe8fc0d153c323a044404dd5d348bd9f55848c639c949fa5b4883b  
8e12c354939cd4a07548e0807e9979eb7a3bbbed00  
A<< (0014+2) (135ms) 990290008e08d7435afb6b4e06d4 9000  
Card MAC: d7435afb6b4e06d4  
Response MAC payload: 4 99029000  
Response MAC: d7435afb6b4e06d4  
ResponseAPDU: 9000  
Original APDU: 8c0704c56f0cbc5a3aa39aabc90dc69db5fef4ec60f1d873619e577e2417d61ca  
87c2226303e10572df37be755acec739ed22611e736585e37e74263c318905c6d3e4a39dfbab8d90  
b8811a1a0c270eb891e273dff22b8d553dc8dbf1314a92adfa29524931d0d979fb68604042baada9  
8a6460c  
APDU payload: 0cbc5a3aa39aabc90dc69db5fef4ec60f1d873619e577e2417d61ca87c2226303  
e10572df37be755acec739ed22611e736585e37e74263c318905c6d3e4a39dfbab8d90b8811a1a0c  
270eb891e273dff22b8d553dc8dbf1314a92adfa29524931d0d979fb68604042baada98a6460c  
Crypt payload: 877101e6a4c40b4b6768c77d8efbbda2d089fea694a22e319e46783aa029232fa  
dccbc56db57ffbed74c4bac38bd0bb01724684d939a57a4b76f8cdf4cbb981ed3dd3c86bca8cd5ad  
237da5822e8d1a30dad8931bb123659bc4e747fea8107d418feccb63adaa14a03fc287b71a7cf8a6  
88c9c  
Verified APDU: 0cbc5a3aa39aabc90dc69db5fef4ec60f1d873619e577e2417d61ca87c2226303  
e10572df37be755acec739ed22611e736585e37e74263c318905c6d3e4a39dfbab8d90b8811a1a0c



```
270eb891e273dff22b8d553dc8dbf1314a92adfa29524931d0d979fb68604042baada98a6460c80
SSC+1: 5d158d52ffb5dd90
MAC payload: 123 8c0704c580000000877101e6a4c40b4b6768c77d8efbbda2d089fea694a22e
319e46783aa029232fadccbc56db57ffbed74c4bac38bd0bb01724684d939a57a4b76f8cdf4cbb98
1ed3dd3c86bca8cd5ad237da5822e8d1a30dad8931bb123659bc4e747fea8107d418feccb63adaa1
4a03fc287b71a7cf8a688c9c
MAC: 53ac7d7d63696618
Final APDU: 8c0704c57d877101e6a4c40b4b6768c77d8efbbda2d089fea694a22e319e46783aa0
29232fadccbc56db57ffbed74c4bac38bd0bb01724684d939a57a4b76f8cdf4cbb981ed3dd3c86bc
a8cd5ad237da5822e8d1a30dad8931bb123659bc4e747fea8107d418feccb63adaa14a03fc287b71
a7cf8a688c9c8e0853ac7d7d6369661800
A>> T=1 (4+0125) 8c0704c5 7d 877101e6a4c40b4b6768c77d8efbbda2d089fea694a22e319e4
6783aa029232fadccbc56db57ffbed74c4bac38bd0bb01724684d939a57a4b76f8cdf4cbb981ed3d
d3c86bca8cd5ad237da5822e8d1a30dad8931bb123659bc4e747fea8107d418feccb63adaa14a03f
c287b71a7cf8a688c9c8e0853ac7d7d6369661800
A<< (0014+2) (130ms) 990290008e08a75466cc3e589998 9000
Card MAC: a75466cc3e589998
Response MAC payload: 4 99029000
Response MAC: a75466cc3e589998
ResponseAPDU: 9000
Original APDU: 8c070534126bbf7a9bb76f93219efd1b34150dc500e5
APDU payload: 6bbf7a9bb76f93219efd1b34150dc500e5
Crypt payload: 871901b5a72a66bb9af15686c0fb9385b14aeaa8ccc8fb03206d7c
Verified APDU: 6bbf7a9bb76f93219efd1b34150dc500e5800000000000
SSC+1: 5d158d52ffb5dd92
MAC payload: 35 8c07053480000000871901b5a72a66bb9af15686c0fb9385b14aeaa8ccc8fb0
3206d7c
MAC: 3e46e10484dabe60
Final APDU: 8c07053425871901b5a72a66bb9af15686c0fb9385b14aeaa8ccc8fb03206d7c8e08
3e46e10484dabe6000
A>> T=1 (4+0037) 8c070534 25 871901b5a72a66bb9af15686c0fb9385b14aeaa8ccc8fb03206
d7c8e083e46e10484dabe6000
A<< (0014+2) (85ms) 990290008e08613f2fc98537ca22 9000
Card MAC: 613f2fc98537ca22
Response MAC payload: 4 99029000
Response MAC: 613f2fc98537ca22
ResponseAPDU: 9000
```

### **Replacing of Signing Certificate**

```
// Cardholder CMK_CERT: b37f3936658cdb2a45ccfe46debc3a63
// Verify PIN1
A>> T=1 (4+0004) 00200001 04 31323334
A<< (0000+2) (110ms) 9000

// Mutual Authenticate with cardholder CMK_CERT
A>> T=1 (4+0000) 00840000 00
A<< (0008+2) (20ms) 2f78e9b77f09fb24 9000
RND.ICC: 2f78e9b77f09fb24
RND.IFD: 1c3e2495fb2dd595
Payload: 1c3e2495fb2dd5952f78e9b77f09fb244ba0e576b8a73750ccfea054febc9d1d6b9e79e
542f6f48868b05412c1f7f1dc
Crypted: a8e02787fe3be4132b1e3102c9a7ea1c93b23d13cc2d918c884e29fd8d1c1a342916a63
```





```
aal6ad94f090af9ed2d30450f
A>> T=1 (4+0048) 00820003 30 a8e02787fe3be4132b1e3102c9a7ea1c93b23d13cc2d918c884
e29fd8d1c1a342916a63aal6ad94f090af9ed2d30450f30
A<< (0048+2) (166ms) 3cf557607621245d2213b56b6041b782aa1def8dba0fc276df8232b2694
c5078eef813433cb6128967f66270dfac570f 9000
Encrypted: 3cf557607621245d2213b56b6041b782aa1def8dba0fc276df8232b2694c5078eef81
3433cb6128967f66270dfac570f
Decrypted: 2f78e9b77f09fb241c3e2495fb2dd595e58e33e467cbd75f055f5e90fbf042452f8d8
49277189c16ffd4cad2ae42d909
K.ICC: e58e33e467cbd75f055f5e90fbf042452f8d849277189c16ffd4cad2ae42d909
K.IFD: 4ba0e576b8a73750ccfea054febcb9d1d6b9e79e542f6f48868b05412c1f7f1dc
K.XOR: ae2ed692df6ce00fc9a1fec4054cdf584413fd7735ee689e97649ec06fb528d5
SK1: ae2ed692df6ce00fc9a1fec4054cdf58
SK1: ae2fd692df6de00ec8a1fec4044cdf58
SK2: 4413fd7735ee689e97649ec06fb528d5
SK2: 4513fd7634ef689e97649ec16eb529d5
SSC: fb2dd5957f09fb24
## Mutual Authentication successful ##
```

```
Original APDU: 8c0780006f308204fd308203e5a0030201020210195adf6f0640288e5609395d3
6fd6783300d06092a864886f70d01010b0500306c310b30090603550406130245453122302006035
5040a0c19415320536572746966697473656572696d69736b6573311f301d06035504030c1
6544553
APDU payload: 308204fd308203e5a0030201020210195adf6f0640288e5609395d36fd6783300
d06092a864886f70d01010b0500306c310b300906035504061302454531223020060355040a0c194
15320536572746966697473656572696d69736b6573311f301d06035504030c16544553
Crypt payload: 8771014c035ad7148ddf50da3f8b6153e7f00d4a9412483ab865efc5bec72ac3f
c8ae15619c6660debe5559233d67d5a7f65ea8cdaabf3130f069a58445fe4e0de828792b83ae0de9
33c22f1af1e8ed97c7c4c6e07e3cb7ca16d81b9602677b598898997ff45eb4451f71687309c4d103
18437
Verified APDU: 308204fd308203e5a0030201020210195adf6f0640288e5609395d36fd6783300
d06092a864886f70d01010b0500306c310b300906035504061302454531223020060355040a0c194
15320536572746966697473656572696d69736b6573311f301d06035504030c1654455380
SSC+1: fb2dd5957f09fb25
MAC payload: 123 8c078000800000008771014c035ad7148ddf50da3f8b6153e7f00d4a941248
3ab865efc5bec72ac3fc8ae15619c6660debe5559233d67d5a7f65ea8cdaabf3130f069a58445fe4
e0de828792b83ae0de933c22f1af1e8ed97c7c4c6e07e3cb7ca16d81b9602677b598898997ff45eb
4451f71687309c4d10318437
MAC: 0fc2edccb108bfe7
Final APDU: 8c0780007d8771014c035ad7148ddf50da3f8b6153e7f00d4a9412483ab865efc5be
c72ac3fc8ae15619c6660debe5559233d67d5a7f65ea8cdaabf3130f069a58445fe4e0de828792b8
3ae0de933c22f1af1e8ed97c7c4c6e07e3cb7ca16d81b9602677b598898997ff45eb4451f7168730
9c4d103184378e080fc2edccb108bfe700
A>> T=1 (4+0125) 8c078000 7d 8771014c035ad7148ddf50da3f8b6153e7f00d4a9412483ab86
5efc5bec72ac3fc8ae15619c6660debe5559233d67d5a7f65ea8cdaabf3130f069a58445fe4e0de8
28792b83ae0de933c22f1af1e8ed97c7c4c6e07e3cb7ca16d81b9602677b598898997ff45eb4451f
71687309c4d103184378e080fc2edccb108bfe700
A<< (0014+2) (179ms) 990290008e08966da2c33114dfed 9000
Card MAC: 966da2c33114dfed
Response MAC payload: 4 99029000
Response MAC: 966da2c33114dfed
ResponseAPDU: 9000
```





Original APDU: 8c07806f6f54206f66204553544549442d534b20323031313118301606092a864886f70d0109011609706b6940736b2e6565301e170d3135303932383132353830355a170d32303039323230353935395a30819e310b3009060355040613024545310f300d060355040a0c0645535445494431

APDU payload: 54206f66204553544549442d534b20323031313118301606092a864886f70d0109011609706b6940736b2e6565301e170d3135303932383132353830355a170d32303039323230353935395a30819e310b3009060355040613024545310f300d060355040a0c0645535445494431

Crypt payload: 8771012034b80c3384671b2137d86a89c7c13c7b66f5611a813e792e88834f6c4e6e38d28b6c5eb99e471f2fd95f1d4c61d5f89bfa7ea8993ede4ce3f02390f7960dd2490781528acda9deb7f567a6ce57eda0f851305ab8dbb2c361350ea02f6ed59139b3ceab2492ea6d25b15dc54a06af76

Verified APDU: 54206f66204553544549442d534b20323031313118301606092a864886f70d0109011609706b6940736b2e6565301e170d3135303932383132353830355a170d32303039323230353935395a30819e310b3009060355040613024545310f300d060355040a0c064553544549443180

SSC+1: fb2dd5957f09fb27

MAC payload: 123 8c07806f800000008771012034b80c3384671b2137d86a89c7c13c7b66f5611a813e792e88834f6c4e6e38d28b6c5eb99e471f2fd95f1d4c61d5f89bfa7ea8993ede4ce3f02390f7960dd2490781528acda9deb7f567a6ce57eda0f851305ab8dbb2c361350ea02f6ed59139b3ceab2492ea6d25b15dc54a06af76

MAC: 7fcbb771c23490eb

Final APDU: 8c07806f7d8771012034b80c3384671b2137d86a89c7c13c7b66f5611a813e792e88834f6c4e6e38d28b6c5eb99e471f2fd95f1d4c61d5f89bfa7ea8993ede4ce3f02390f7960dd2490781528acda9deb7f567a6ce57eda0f851305ab8dbb2c361350ea02f6ed59139b3ceab2492ea6d25b15dc54a06af768e087fcbb771c23490eb00

A>> T=1 (4+0125) 8c07806f 7d 8771012034b80c3384671b2137d86a89c7c13c7b66f5611a813e792e88834f6c4e6e38d28b6c5eb99e471f2fd95f1d4c61d5f89bfa7ea8993ede4ce3f02390f7960dd2490781528acda9deb7f567a6ce57eda0f851305ab8dbb2c361350ea02f6ed59139b3ceab2492ea6d25b15dc54a06af768e087fcbb771c23490eb00

A<< (0014+2) (118ms) 990290008e08f435cb23f2793011 9000

Card MAC: f435cb23f2793011

Response MAC payload: 4 99029000

Response MAC: f435cb23f2793011

ResponseAPDU: 9000

Original APDU: 8c0780de6f1a3018060355040b0c116469676974616c207369676e61747572653126302406035504030c1d4dc3844e4e494b2c4d4152492d4c4949532c343731303130313030333333110300e06035504040c074dc3844e4e494b31123010060355042a0c094d4152492d4c494953311430120603

APDU payload: 1a3018060355040b0c116469676974616c207369676e61747572653126302406035504030c1d4dc3844e4e494b2c4d4152492d4c4949532c343731303130313030333333110300e06035504040c074dc3844e4e494b31123010060355042a0c094d4152492d4c494953311430120603

Crypt payload: 877101da430cb478f709d14ac3386a42a5ce0b857f238354ce799a55bee271dba2afefd8e0b722d79942ccfba40f31824cc961433d4c5f279a00c4109069bfac3045179ddc937d6fc408138b4ba2b21de751e81b16a587d84c891d1293326b2125d8bc2533fa37fe840c99c12291935c6e4646

Verified APDU: 1a3018060355040b0c116469676974616c207369676e61747572653126302406035504030c1d4dc3844e4e494b2c4d4152492d4c4949532c343731303130313030333333110300e06035504040c074dc3844e4e494b31123010060355042a0c094d4152492d4c49495331143012060380

SSC+1: fb2dd5957f09fb29

MAC payload: 123 8c0780de80000000877101da430cb478f709d14ac3386a42a5ce0b857f238354ce799a55bee271dba2afefd8e0b722d79942ccfba40f31824cc961433d4c5f279a00c4109069bfac3045179ddc937d6fc408138b4ba2b21de751e81b16a587d84c891d1293326b2125d8bc2533fa37fe840c99c12291935c6e4646



MAC: 08c22ff3d23b3b1c  
Final APDU: 8c0780de7d877101da430cb478f709d14ac3386a42a5ce0b857f238354ce799a55bee271dba2afefd8e0b722d79942ccfba40f31824cc961433d4c5f279a00c4109069bfac3045179ddc937d6fc408138b4ba2b21de751e81b16a587d84c891d1293326b2125d8bc2533fa37fe840c99c12291935c6e46468e0808c22ff3d23b3b1c00  
A>> T=1 (4+0125) 8c0780de 7d 877101da430cb478f709d14ac3386a42a5ce0b857f238354ce799a55bee271dba2afefd8e0b722d79942ccfba40f31824cc961433d4c5f279a00c4109069bfac3045179ddc937d6fc408138b4ba2b21de751e81b16a587d84c891d1293326b2125d8bc2533fa37fe840c99c12291935c6e46468e0808c22ff3d23b3b1c00  
A<< (0014+2) (130ms) 990290008e08f54fded361761ad2 9000  
Card MAC: f54fded361761ad2  
Response MAC payload: 4 99029000  
Response MAC: f54fded361761ad2  
ResponseAPDU: 9000  
Original APDU: 8c07814d6f550405130b343731303130313030333330820122300d06092a864886f70d01010105000382010f003082010a0282010100907ed9071183ef2b5a1974265319de728a3db03d9269bb8ce3f2ec8b862052adf41599055a303bac4c24c14a97ad428abe29a2f96f5b19b9228b2d366b24  
APDU payload: 550405130b343731303130313030333330820122300d06092a864886f70d01010105000382010f003082010a0282010100907ed9071183ef2b5a1974265319de728a3db03d9269bb8ce3f2ec8b862052adf41599055a303bac4c24c14a97ad428abe29a2f96f5b19b9228b2d366b24  
Crypt payload: 8771011d5a6b98a0d105512f72f90ac7786fcb80c7b223414a925924cddf7885e7ec10b58908d316f6e9382d966b7ec7c2800d8b9b1a7a4d8bb96502946a83c8e1be71d19853a6c10c7bfff468eaf51530407bd11289f615a1e2391c3c9b48293b37485acd3dd99be4d4c1522db098c4a3ec3be  
Verified APDU: 550405130b343731303130313030333330820122300d06092a864886f70d01010105000382010f003082010a0282010100907ed9071183ef2b5a1974265319de728a3db03d9269bb8ce3f2ec8b862052adf41599055a303bac4c24c14a97ad428abe29a2f96f5b19b9228b2d366b2480  
SSC+1: fb2dd5957f09fb2b  
MAC payload: 123 8c07814d800000008771011d5a6b98a0d105512f72f90ac7786fcb80c7b223414a925924cddf7885e7ec10b58908d316f6e9382d966b7ec7c2800d8b9b1a7a4d8bb96502946a83c8e1be71d19853a6c10c7bfff468eaf51530407bd11289f615a1e2391c3c9b48293b37485acd3dd99be4d4c1522db098c4a3ec3be  
MAC: c656d6fe5930aee8  
Final APDU: 8c07814d7d8771011d5a6b98a0d105512f72f90ac7786fcb80c7b223414a925924cddf7885e7ec10b58908d316f6e9382d966b7ec7c2800d8b9b1a7a4d8bb96502946a83c8e1be71d19853a6c10c7bfff468eaf51530407bd11289f615a1e2391c3c9b48293b37485acd3dd99be4d4c1522db098c4a3ec3be8e08c656d6fe5930aee800  
A>> T=1 (4+0125) 8c07814d 7d 8771011d5a6b98a0d105512f72f90ac7786fcb80c7b223414a925924cddf7885e7ec10b58908d316f6e9382d966b7ec7c2800d8b9b1a7a4d8bb96502946a83c8e1be71d19853a6c10c7bfff468eaf51530407bd11289f615a1e2391c3c9b48293b37485acd3dd99be4d4c1522db098c4a3ec3be8e08c656d6fe5930aee800  
A<< (0014+2) (130ms) 990290008e082a5cfee3860f389b 9000  
Card MAC: 2a5cfee3860f389b  
Response MAC payload: 4 99029000  
Response MAC: 2a5cfee3860f389b  
ResponseAPDU: 9000  
Original APDU: 8c0781bc6f51c5929008ec3286b3ec05226660c722045c0fa2d76884e21e07f7bebc29784076db6f705884160466c7c6cc47d50905e28c4f00388b532558263dfd39ceb8b659de27f3ac09146c10179d176f1eca27e1cfa850d1ad694420fd70da229be11ddd4d9d5af28355689203872f5606ed  
APDU payload: 51c5929008ec3286b3ec05226660c722045c0fa2d76884e21e07f7bebc2978407



6db6f705884160466c7c6cc47d50905e28c4f00388b532558263dfd39ceb8b659de27f3ac09146c1  
0179d176f1eca27e1cfa850d1ad694420fd70da229be11ddd4d9d5af28355689203872f5606ed  
Crypt payload: 8771019b2f287d90a586c048542fb2c37b49cc85b008d31a32382f67690cf6028  
71cc123c7f97828affc1c501646118ea30c8d5ba4d3072e5cf1bf3d9b13affff364e1132ea57859d  
88622186861466c4045ade041e08e2de98f57f17645c4086c5cf130f4c330e9de459eda22ada6110  
efb67  
Verified APDU: 51c5929008ec3286b3ec05226660c722045c0fa2d76884e21e07f77bebc2978407  
6db6f705884160466c7c6cc47d50905e28c4f00388b532558263dfd39ceb8b659de27f3ac09146c1  
0179d176f1eca27e1cfa850d1ad694420fd70da229be11ddd4d9d5af28355689203872f5606ed80  
SSC+1: fb2dd5957f09fb2d  
MAC payload: 123 8c0781bc800000008771019b2f287d90a586c048542fb2c37b49cc85b008d3  
1a32382f67690cf602871cc123c7f97828affc1c501646118ea30c8d5ba4d3072e5cf1bf3d9b13af  
fff364e1132ea57859d88622186861466c4045ade041e08e2de98f57f17645c4086c5cf130f4c330  
e9de459eda22ada6110efb67  
MAC: 7c5f45544388505f  
Final APDU: 8c0781bc7d8771019b2f287d90a586c048542fb2c37b49cc85b008d31a32382f6769  
0cf602871cc123c7f97828affc1c501646118ea30c8d5ba4d3072e5cf1bf3d9b13affff364e1132e  
a57859d88622186861466c4045ade041e08e2de98f57f17645c4086c5cf130f4c330e9de459eda22  
ada6110efb678e087c5f45544388505f00  
A>> T=1 (4+0125) 8c0781bc 7d 8771019b2f287d90a586c048542fb2c37b49cc85b008d31a323  
82f67690cf602871cc123c7f97828affc1c501646118ea30c8d5ba4d3072e5cf1bf3d9b13affff36  
4e1132ea57859d88622186861466c4045ade041e08e2de98f57f17645c4086c5cf130f4c330e9de4  
59eda22ada6110efb678e087c5f45544388505f00  
A<< (0014+2) (128ms) 990290008e08b012304685509b91 9000  
Card MAC: b012304685509b91  
Response MAC payload: 4 99029000  
Response MAC: b012304685509b91  
ResponseAPDU: 9000  
Original APDU: 8c07822b6f616b2946072a54dc887f2060190327b29f90eb349957d2bec0d3a02  
46f2f311ec4bc90f5b2efec6a182979a87ec222f7cac5ea39dd2b9e32ee1aac93e0186b89d1452ea  
36032f918cce637487d1f80eff816fd0203010001a38201663082016230090603551d13040230003  
00e0603  
APDU payload: 616b2946072a54dc887f2060190327b29f90eb349957d2bec0d3a0246f2f311ec  
4bc90f5b2efec6a182979a87ec222f7cac5ea39dd2b9e32ee1aac93e0186b89d1452ea36032f918c  
ce637487d1f80eff816fd0203010001a38201663082016230090603551d1304023000300e0603  
Crypt payload: 8771013a57609145483e83f33faee0be184a5a8c190a14ebf491fb5cbfd3379c2  
6419c8d2dc1f8e10a23c086f61f89b6865920777711d8fcb1fc38be90d93457c6f89926f34a3bdbc  
85892ccc7f575bd3ad376a61ac20da5700428bf5b2690442c5469a9932ae9d3b2a0a10ab28e0b049  
7706d  
Verified APDU: 616b2946072a54dc887f2060190327b29f90eb349957d2bec0d3a0246f2f311ec  
4bc90f5b2efec6a182979a87ec222f7cac5ea39dd2b9e32ee1aac93e0186b89d1452ea36032f918c  
ce637487d1f80eff816fd0203010001a38201663082016230090603551d1304023000300e060380  
SSC+1: fb2dd5957f09fb2f  
MAC payload: 123 8c07822b800000008771013a57609145483e83f33faee0be184a5a8c190a14  
ebf491fb5cbfd3379c26419c8d2dc1f8e10a23c086f61f89b6865920777711d8fcb1fc38be90d934  
57c6f89926f34a3bdbc85892ccc7f575bd3ad376a61ac20da5700428bf5b2690442c5469a9932ae9  
d3b2a0a10ab28e0b0497706d  
MAC: 84cbe1047dc41cd1  
Final APDU: 8c07822b7d8771013a57609145483e83f33faee0be184a5a8c190a14ebf491fb5cbf  
d3379c26419c8d2dc1f8e10a23c086f61f89b6865920777711d8fcb1fc38be90d93457c6f89926f3  
4a3bdbc85892ccc7f575bd3ad376a61ac20da5700428bf5b2690442c5469a9932ae9d3b2a0a10ab2  
8e0b0497706d8e0884cbe1047dc41cd100



```
A>> T=1 (4+0125) 8c07822b 7d 8771013a57609145483e83f33faee0be184a5a8c190a14ebf49
1fb5cbfd3379c26419c8d2dc1f8e10a23c086f61f89b6865920777711d8fcb1fc38be90d93457c6f
89926f34a3bdbc85892ccc7f575bd3ad376a61ac20da5700428bf5b2690442c5469a9932ae9d3b2a
0a10ab28e0b0497706d8e0884cbe1047dc41cd100
A<< (0014+2) (120ms) 990290008e08fc79359ddf034736 9000
Card MAC: fc79359ddf034736
Response MAC payload: 4 99029000
Response MAC: fc79359ddf034736
ResponseAPDU: 9000
Original APDU: 8c07829a6f551d0f0101ff0404030206403081990603551d2004819130818e308
18b060a2b06010401ce1f030101307d305806082b06010505070202304c1e4a00410069006e00750
06c0074002000740065007300740069006d006900730065006b0073002e0020004f006e006c00790
0200066
APDU payload: 551d0f0101ff0404030206403081990603551d2004819130818e30818b060a2b0
6010401ce1f030101307d305806082b06010505070202304c1e4a00410069006e0075006c0074002
000740065007300740069006d006900730065006b0073002e0020004f006e006c007900200066
Crypt payload: 8771010f8f2afa615e5c4686abdb6f13029dfa445dc17c4c0452db7886207e5b9
81f240d5f9f4729f8d2609ee34a19563cca31a91d58a371593c32d32dbe9bfbfb30c6d28d0e09b5
32c38334ae90fcee4ef230d4d8ba4e3a140f0a2bf059cef29de42c68c7b036bd89c61296e0af59dc
e2557
Verified APDU: 551d0f0101ff0404030206403081990603551d2004819130818e30818b060a2b0
6010401ce1f030101307d305806082b06010505070202304c1e4a00410069006e0075006c0074002
000740065007300740069006d006900730065006b0073002e0020004f006e006c00790020006680
SSC+1: fb2dd5957f09fb31
MAC payload: 123 8c07829a800000008771010f8f2afa615e5c4686abdb6f13029dfa445dc17c
4c0452db7886207e5b981f240d5f9f4729f8d2609ee34a19563cca31a91d58a371593c32d32dbe9b
fbfb30c6d28d0e09b532c38334ae90fcee4ef230d4d8ba4e3a140f0a2bf059cef29de42c68c7b03
6bd89c61296e0af59dce2557
MAC: ed53a1adc8689393
Final APDU: 8c07829a7d8771010f8f2afa615e5c4686abdb6f13029dfa445dc17c4c0452db7886
207e5b981f240d5f9f4729f8d2609ee34a19563cca31a91d58a371593c32d32dbe9bfbfb30c6d28
d0e09b532c38334ae90fcee4ef230d4d8ba4e3a140f0a2bf059cef29de42c68c7b036bd89c61296e
0af59dce25578e08ed53a1adc868939300
A>> T=1 (4+0125) 8c07829a 7d 8771010f8f2afa615e5c4686abdb6f13029dfa445dc17c4c045
2db7886207e5b981f240d5f9f4729f8d2609ee34a19563cca31a91d58a371593c32d32dbe9bfbfb30c
6d28d0e09b532c38334ae90fcee4ef230d4d8ba4e3a140f0a2bf059cef29de42c68c7b036bd89c
61296e0af59dce25578e08ed53a1adc868939300
A<< (0014+2) (130ms) 990290008e08d2bb699e8e9db65f 9000
Card MAC: d2bb699e8e9db65f
Response MAC payload: 4 99029000
Response MAC: d2bb699e8e9db65f
ResponseAPDU: 9000
Original APDU: 8c0783096f006f0072002000740065007300740069006e0067002e302106082b0
60105050702011615687474703a2f2f777772e736b2e65652f6370732f301d0603551d0e0416041
4b9f44afc992357b3f4c2fdcd128d09ea1293943302206082b06010505070103041630143008060
604008e
APDU payload: 006f0072002000740065007300740069006e0067002e302106082b06010505070
2011615687474703a2f2f777772e736b2e65652f6370732f301d0603551d0e04160414b9f44afc9
92357b3f4c2fdcd128d09ea1293943302206082b06010505070103041630143008060604008e
Crypt payload: 87710197cf09a7dba8645835a49926f2470f1bc96ee46becd4adac75d1d3c9bbc
e54cef4db43d7850b7cfd0c1558f5d4a299b1a318caf3583c0d1afb7d207e3b80e6ccc954883549
28bbe2ceb854c91d694f2e0b2b81a6e1b336a8948417ef8639b27ff158da9c50007e3e7c346b6552
```



33fa5  
Verified APDU: 006f0072002000740065007300740069006e0067002e302106082b06010505070  
2011615687474703a2f2f777772e736b2e65652f6370732f301d0603551d0e04160414b9f44afc9  
92357b3f4c2fdcd128d09ea1293943302206082b06010505070103041630143008060604008e80  
SSC+1: fb2dd5957f09fb33  
MAC payload: 123 8c078309800000087710197cf09a7dba8645835a49926f2470f1bc96ee46b  
ecd4adac75d1d3c9bbce54cef4db43d7850b7cfd0c1558f5d4a299b1a318caf3583c0d1afbd7d207  
e3b80e6ccc95488354928bbe2ceb854c91d694f2e0b2b81a6e1b336a8948417ef8639b27ff158da9  
c50007e3e7c346b655233fa5  
MAC: 30b7145eaae78c82  
Final APDU: 8c0783097d87710197cf09a7dba8645835a49926f2470f1bc96ee46becd4adac75d1  
d3c9bbce54cef4db43d7850b7cfd0c1558f5d4a299b1a318caf3583c0d1afbd7d207e3b80e6ccc95  
488354928bbe2ceb854c91d694f2e0b2b81a6e1b336a8948417ef8639b27ff158da9c50007e3e7c3  
46b655233fa58e0830b7145eaae78c8200  
A>> T=1 (4+0125) 8c078309 7d 87710197cf09a7dba8645835a49926f2470f1bc96ee46becd4a  
dac75d1d3c9bbce54cef4db43d7850b7cfd0c1558f5d4a299b1a318caf3583c0d1afbd7d207e3b80  
e6ccc95488354928bbe2ceb854c91d694f2e0b2b81a6e1b336a8948417ef8639b27ff158da9c5000  
7e3e7c346b655233fa58e0830b7145eaae78c8200  
A<< (0014+2) (120ms) 990290008e08303670e9b29b3a01 9000  
Card MAC: 303670e9b29b3a01  
Response MAC payload: 4 99029000  
Response MAC: 303670e9b29b3a01  
ResponseAPDU: 9000  
Original APDU: 8c0783786f4601013008060604008e460104301f0603551d2304183016801441b  
6fec5b1b1b453138cfafa62d0346d6d22340a30450603551d1f043e303c303aa038a036863468747  
4703a2f2f777772e736b2e65652f7265706f7369746f72792f63726c732f746573745f657374656  
9643230  
APDU payload: 4601013008060604008e460104301f0603551d2304183016801441b6fec5b1b1b  
453138cfafa62d0346d6d22340a30450603551d1f043e303c303aa038a0368634687474703a2f2f7  
777772e736b2e65652f7265706f7369746f72792f63726c732f746573745f6573746569643230  
Crypt payload: 87710186a8bfa12f8488de615c3ed775c34dd3c15dc39bec7af35b937df5f5ff6  
002acc09ca9df2e4a227b7023f5a9ce76eb0d4c3fa11b18ad61835e176d22bfc93a90f95ae55b86  
7b656c71e9250ac131fa75afeeda7f0ce20e7ba0d26c609021b20548871f75e7d905f62971048a88  
d77a6  
Verified APDU: 4601013008060604008e460104301f0603551d2304183016801441b6fec5b1b1b  
453138cfafa62d0346d6d22340a30450603551d1f043e303c303aa038a0368634687474703a2f2f7  
777772e736b2e65652f7265706f7369746f72792f63726c732f746573745f657374656964323080  
SSC+1: fb2dd5957f09fb35  
MAC payload: 123 8c078378800000087710186a8bfa12f8488de615c3ed775c34dd3c15dc39b  
ec7af35b937df5f5ff6002acc09ca9df2e4a227b7023f5a9ce76eb0d4c3fa11b18ad61835e176d22  
bfc93a90f95ae55b867b656c71e9250ac131fa75afeeda7f0ce20e7ba0d26c609021b20548871f7  
5e7d905f62971048a88d77a6  
MAC: 82b632f62a4c1f63  
Final APDU: 8c0783787d87710186a8bfa12f8488de615c3ed775c34dd3c15dc39bec7af35b937d  
f5f5ff6002acc09ca9df2e4a227b7023f5a9ce76eb0d4c3fa11b18ad61835e176d22bfc93a90f95  
ae55b867b656c71e9250ac131fa75afeeda7f0ce20e7ba0d26c609021b20548871f75e7d905f6297  
1048a88d77a68e0882b632f62a4c1f6300  
A>> T=1 (4+0125) 8c078378 7d 87710186a8bfa12f8488de615c3ed775c34dd3c15dc39bec7af  
35b937df5f5ff6002acc09ca9df2e4a227b7023f5a9ce76eb0d4c3fa11b18ad61835e176d22bfc9  
3a90f95ae55b867b656c71e9250ac131fa75afeeda7f0ce20e7ba0d26c609021b20548871f75e7d9  
05f62971048a88d77a68e0882b632f62a4c1f6300  
A<< (0014+2) (130ms) 990290008e0839278ae23a059633 9000



## TB-SPEC-EstEID-Chip-App-v3.5-20170314 Politsei- ja Piirivalveamet

---

Card MAC: 39278ae23a059633  
Response MAC payload: 4 99029000  
Response MAC: 39278ae23a059633  
ResponseAPDU: 9000  
Original APDU: 8c0783e76f31312e63726c300d06092a864886f70d01010b05000382010100180a465a038c9c2193acfd3d46d1f99e60f2225fb12f6ae9e820e35546702cb9b3df7bda3307e0088327643e331cd0d73801cb8eae146df87494d3de3de0fe2f8cd548386fa1dda56ad2a74d56aa29e77e5f5122  
APDU payload: 31312e63726c300d06092a864886f70d01010b05000382010100180a465a038c9c2193acfd3d46d1f99e60f2225fb12f6ae9e820e35546702cb9b3df7bda3307e0088327643e331cd0d73801cb8eae146df87494d3de3de0fe2f8cd548386fa1dda56ad2a74d56aa29e77e5f5122  
Crypt payload: 8771011a4b10ff18aed53bb0b74a190fe14419074c0cbd2d744887d27270555d079253fcb82241bd14dc6690711971f8632e78d188063612358b2795c81d1aabe2c0ab0ae3c11f9962dc3ba620e55624822e177f75073a53988c5067b3008a0e9f22a6408c6ef9f2a007f34948dc2523586bc9  
Verified APDU: 31312e63726c300d06092a864886f70d01010b05000382010100180a465a038c9c2193acfd3d46d1f99e60f2225fb12f6ae9e820e35546702cb9b3df7bda3307e0088327643e331cd0d73801cb8eae146df87494d3de3de0fe2f8cd548386fa1dda56ad2a74d56aa29e77e5f512280SSC+1: fb2dd5957f09fb37  
MAC payload: 123 8c0783e7800000008771011a4b10ff18aed53bb0b74a190fe14419074c0cbd2d744887d27270555d079253fcb82241bd14dc6690711971f8632e78d188063612358b2795c81d1aabe2c0ab0ae3c11f9962dc3ba620e55624822e177f75073a53988c5067b3008a0e9f22a6408c6ef9f2a007f34948dc2523586bc9  
MAC: 83fabbe6e6122660  
Final APDU: 8c0783e77d8771011a4b10ff18aed53bb0b74a190fe14419074c0cbd2d744887d27270555d079253fcb82241bd14dc6690711971f8632e78d188063612358b2795c81d1aabe2c0ab0ae3c11f9962dc3ba620e55624822e177f75073a53988c5067b3008a0e9f22a6408c6ef9f2a007f34948dc2523586bc98e0883fabbe6e612266000  
A>> T=1 (4+0125) 8c0783e7 7d 8771011a4b10ff18aed53bb0b74a190fe14419074c0cbd2d744887d27270555d079253fcb82241bd14dc6690711971f8632e78d188063612358b2795c81d1aabe2c0ab0ae3c11f9962dc3ba620e55624822e177f75073a53988c5067b3008a0e9f22a6408c6ef9f2a007f34948dc2523586bc98e0883fabbe6e612266000  
A<< (0014+2) (130ms) 990290008e08109771268ec4bee6 9000  
Card MAC: 109771268ec4bee6  
Response MAC payload: 4 99029000  
Response MAC: 109771268ec4bee6  
ResponseAPDU: 9000  
Original APDU: 8c0784566f6aa2afdf3e85e619b81e547322cd61d1ff1ee9569ad3db5ca656b7a84262bf306cc4bd0bc8e77d106b0546d96b32d7ea0bb520a6dc7b401b5d58ea8947e81f324c2af567134a1c6705a0a57aa30fe25a015a5a129a1a8e0a90bc820e223a6ef6a17046ddf97879b37ef4c07f2c3504  
APDU payload: 6aa2afdf3e85e619b81e547322cd61d1ff1ee9569ad3db5ca656b7a84262bf306cc4bd0bc8e77d106b0546d96b32d7ea0bb520a6dc7b401b5d58ea8947e81f324c2af567134a1c6705a0a57aa30fe25a015a5a129a1a8e0a90bc820e223a6ef6a17046ddf97879b37ef4c07f2c3504  
Crypt payload: 877101ff21172a396d09a090f35e8ca38577cf6981fa031cb2449849a861fbb02a1a0933451506cff9edfebf13db9df539d1e4fb7fdffdcf6553e1203f971eac1b22b843f4f9fa2cc043a7320a919ab78d71738141a07c08f155038d691b83fda6e11eafb07b7442728b1fb8bf1936c8a64ffd  
Verified APDU: 6aa2afdf3e85e619b81e547322cd61d1ff1ee9569ad3db5ca656b7a84262bf306cc4bd0bc8e77d106b0546d96b32d7ea0bb520a6dc7b401b5d58ea8947e81f324c2af567134a1c6705a0a57aa30fe25a015a5a129a1a8e0a90bc820e223a6ef6a17046ddf97879b37ef4c07f2c350480SSC+1: fb2dd5957f09fb39



## TB-SPEC-EstEID-Chip-App-v3.5-20170314 Politsei- ja Piirivalveamet

---

MAC payload: 123 8c0784568000000877101ff21172a396d09a090f35e8ca38577cf6981fa03  
1cb2449849a861fbb02a1a0933451506cff9edfebf13db9df539d1e4fb7fdffdcf6553e1203f971e  
ac1b22b843f4f9fa2cc043a7320a919ab78d71738141a07c08f155038d691b83fda6e11eafb07b74  
42728b1fb8bf1936c8a64ffd

MAC: 9b0b20a9c1a966cc

Final APDU: 8c0784567d877101ff21172a396d09a090f35e8ca38577cf6981fa031cb2449849a8  
61fbb02a1a0933451506cff9edfebf13db9df539d1e4fb7fdffdcf6553e1203f971eac1b22b843f4  
f9fa2cc043a7320a919ab78d71738141a07c08f155038d691b83fda6e11eafb07b7442728b1fb8bf  
1936c8a64ffd8e089b0b20a9c1a966cc00

A>> T=1 (4+0125) 8c078456 7d 877101ff21172a396d09a090f35e8ca38577cf6981fa031cb24  
49849a861fbb02a1a0933451506cff9edfebf13db9df539d1e4fb7fdffdcf6553e1203f971eac1b2  
2b843f4f9fa2cc043a7320a919ab78d71738141a07c08f155038d691b83fda6e11eafb07b7442728  
b1fb8bf1936c8a64ffd8e089b0b20a9c1a966cc00

A<< (0014+2) (130ms) 990290008e08eb143940c7909139 9000

Card MAC: eb143940c7909139

Response MAC payload: 4 99029000

Response MAC: eb143940c7909139

ResponseAPDU: 9000

Original APDU: 8c0784c53c13a60e1d58ae02a174e96f305a9d368a98ead93f37ee419f26302d4  
053d865105e950b569929a2d82897ec07ba37353bd38677667f2ae868589834e4

APDU payload: 13a60e1d58ae02a174e96f305a9d368a98ead93f37ee419f26302d4053d865105  
e950b569929a2d82897ec07ba37353bd38677667f2ae868589834e4

Crypt payload: 87410184d3bd4a5c29d6a602caccfb74e5d0c3ecda98a7421d9b8d31487ff597f  
df17db1a1fde801f058596f3fcfc3004a0ee9c05c9d5196817fab1a61a9153befe8e9

Verified APDU: 13a60e1d58ae02a174e96f305a9d368a98ead93f37ee419f26302d4053d865105  
e950b569929a2d82897ec07ba37353bd38677667f2ae868589834e480000000

SSC+1: fb2dd5957f09fb3b

MAC payload: 75 8c0784c58000000087410184d3bd4a5c29d6a602caccfb74e5d0c3ecda98a74  
21d9b8d31487ff597fdf17db1a1fde801f058596f3fcfc3004a0ee9c05c9d5196817fab1a61a9153  
befe8e9

MAC: c26ef422c6973c65

Final APDU: 8c0784c54d87410184d3bd4a5c29d6a602caccfb74e5d0c3ecda98a7421d9b8d3148  
7ff597fdf17db1a1fde801f058596f3fcfc3004a0ee9c05c9d5196817fab1a61a9153befe8e98e08  
c26ef422c6973c6500

A>> T=1 (4+0077) 8c0784c5 4d 87410184d3bd4a5c29d6a602caccfb74e5d0c3ecda98a7421d9  
b8d31487ff597fdf17db1a1fde801f058596f3fcfc3004a0ee9c05c9d5196817fab1a61a9153befe  
8e98e08c26ef422c6973c6500

A<< (0014+2) (110ms) 990290008e0829a94b6f63588e54 9000

Card MAC: 29a94b6f63588e54

Response MAC payload: 4 99029000

Response MAC: 29a94b6f63588e54

ResponseAPDU: 9000